Diplomarbeit

Mikroarchitektur-Synthese
mit
genetischen Algorithmen

Markus Lorenz

Diplomarbeit
am Fachbereich Informatik
der Universität Dortmund

27. November 1997

Betreuer:

Dipl.–Inform. Birger Landwehr
Prof. Dr. Peter Marwedel

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Komplexität digitaler Schaltungen hat in den letzten Jahrzehnten stark zugenommen. So werden durch die ständig steigende Miniaturisierung heutzutage mehr als 10^6 Transistoren auf einem Chip integriert (VLSI[1]). Trotz vorhandener physikalischer Schranken ist ein Ende dieser Entwicklung bislang noch nicht absehbar. Jedoch kann die Möglichkeit, immer größere Schaltungen auf einer gegebenen Chipfläche zu plazieren, von den Entwicklern nur unzureichend genutzt werden, da größere Schaltungen auch eine längere Entwicklungs- und Testphase erfordern. Mit dem steigenden Trend zu ASICs[2] besteht weiterhin ein zunehmender Bedarf an der Verkürzung des Design-Zyklus (*Time-To-Market*). Dazu sind Methoden wünschenswert, die eine Automatisierung des Entwurfsprozesses ermöglichen und aufgrund ihrer Konstruktion Fehlerfreiheit garantieren (*Correctness-by-Construction*).

Die Aufteilung eines Entwurfs erfolgt in unterschiedliche Domänen. So werden in der Regel die *Verhaltens-*, *Struktur-* und *geometrische/physikalische* Domäne betrachtet, deren graphische Darstellung von Gajski [GK83] in Form des Y-Diagramms vorgeschlagen wurde. Innerhalb einer Domäne erfolgt hierbei eine Unterteilung in verschiedene Abstraktionsebenen. Die Ebene mit dem größten Abstraktionsgrad stellt die System-Ebene dar, bei der eine Entwurfsbeschreibung durch grundlegende Systemkomponenten beschrieben wird. Ausgehend von der System-Ebene erfolgt im Y-Diagramm eine schrittweise Verfeinerung des Entwurfsprozesses von außen nach innen über die algorithmische, Register-Transfer- und Logik-Ebene, bis mit der Schaltkreis-Ebene der niedrigste Abstraktionsgrad erreicht ist. Der Mittelpunkt des Y-Diagramms stellt letztendlich die konkrete Hardware-Realisierung dar.

1.1 Aufgaben der Mikroarchitektur-Synthese

Generell wird unter *Synthese* der Übergang von der Verhaltens-Domäne zur Struktur-Domäne verstanden. Dabei wird das spezifizierte Verhalten unter Berücksichtigung von Randbedingungen auf eine Struktur abgebildet, die diesem Verhalten entspricht. Eine Einordnung der Mikroarchitektur-Synthese (*High-Level Synthesis*) in den Entwurfsablauf wird in Abbildung 1.1 anhand des Y-Diagramms vorgenommen.

In diesem Fall überführt die Mikroarchitektur-Synthese eine Verhaltensbeschreibung auf der algorithmischen Ebene in eine äquivalente Strukturbeschreibung auf Register-Transfer-Ebene. Die Spezifikation eines Entwurfs in Form einer algorithmischen Verhaltensbeschreibung hat gegenüber einer äquivalenten Strukturbeschreibung den Vorteil, daß sie bei komplexen Entwürfen

[1] VLSI (Very Large Scale Integration)
[2] ASIC (Application Specific Integrated Circuit)

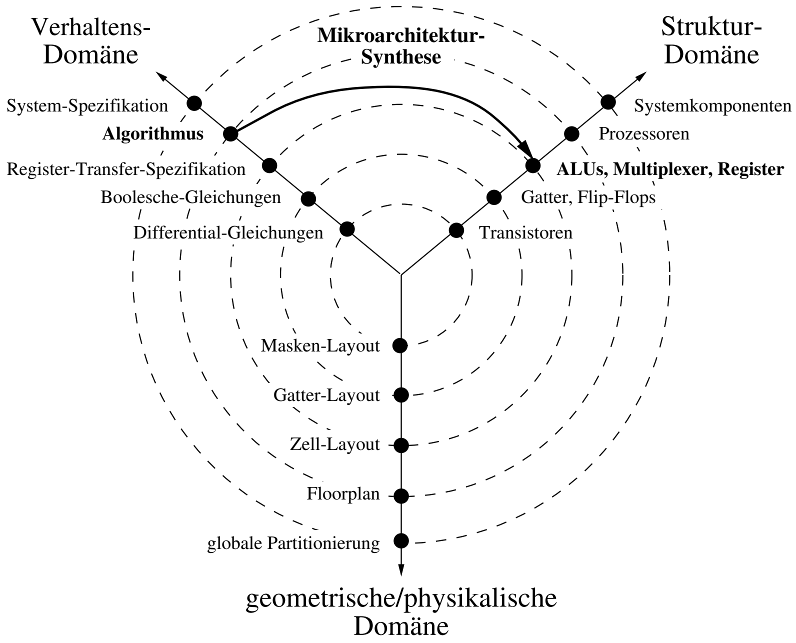


Abbildung 1.1: Einordnung der Mikroarchitektur-Synthese in den Entwurfsprozeß

einfacher zu verstehen und zu ändern ist. Als Beschreibungssprache hat sich in den letzten Jahren VHDL[3] [IEE88] etabliert, mit der eine Beschreibung sowohl auf der Verhaltens-Domäne als auch auf der Struktur-Domäne möglich ist.

Der Ablauf der Mikroarchitektur-Synthese beginnt mit dem Einlesen der algorithmischen Verhaltensbeschreibung des Designs und der Transformation in eine äquivalente interne Darstellung. Hierbei werden vom Entwickler vorgegebene Randbedingungen[4] berücksichtigt. Am häufigsten werden Graphen zur internen Darstellung von Kontroll- und Datenfluß verwendet. Im Falle einer datenflußorientierten Synthese erfolgt hierbei in der Regel eine Darstellung in Form eines separaten Kontroll- und Datenflußgraphen. So wird im Kontrollflußgraphen durch die Beschreibung der Kontrollstrukturen wie *IF*, *CASE* und *LOOP* die Reihenfolge beschrieben, in der die Operationen abgearbeitet werden. Hierbei führt die Darstellung von *LOOP*-Anweisungen in der Regel zu Zyklen im Graphen. Der Datenflußgraph stellt hingegen einen azyklischen Graphen dar, bei dem die Operationen durch Knoten und die Datenabhängigkeiten durch Kanten zwischen den Knoten repräsentiert werden. An dieser Stelle können optional Optimierungen der internen Beschreibungen, wie zum Beispiel algebraische Transformationen, durchgeführt werden.
Der eigentliche Syntheseschritt erfolgt mit der Realisierung von Scheduling, Allokation und Ressourcen-Bindung. Dabei wird jede Operation einem Ausführungszeitpunkt und einem Hardware-Baustein zugewiesen, der eine Komponente der Register-Transfer-Ebene darstellt.
Zum Schluß erfolgt auf der Basis dieser Zuweisungen die Generierung der Register-Transfer-Netzliste und die Erzeugung einer Steuerwerk-Spezifikation. Das Steuerwerk, dargestellt durch einen endlichen Automaten, veranlaßt aufgrund des aktuellen Zustands und der aktuell berechneten Werte des Rechenwerks die Ausführung von Operationen und Register-Transfers.

In Abbildung 1.2 ist die Durchführung der Teilaufgaben *Scheduling*, *Allokation* und *Ressourcen-Bindung* dargestellt.

[3]VHDL (VHSIC (Very High Speed Integrated Circuit) Hardware Description Language)
[4]Das können zum Beispiel minimale oder maximale Zeitabstände zwischen Operationen oder die Berücksichtigung spezieller Bausteinbibliotheken sein.

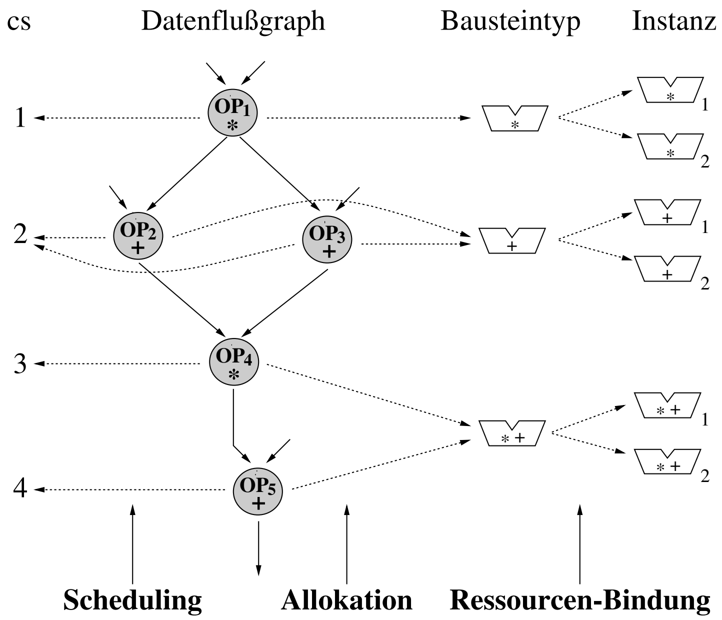


Abbildung 1.2: Aufgaben der Mikroarchitektur-Synthese

Die Durchführung des *Scheduling* hat das Ziel, jeder Operation eines Datenflußgraphen einen Startzeitpunkt in Form eines Kontrollschrittes[5] (*Control-Step* (cs)) zuzuweisen. Mit der *Allokation* erfolgt für jede Operation die Bereitstellung und Selektion eines geeigneten Bausteintyps auf dem die jeweilige Operation ausgeführt werden kann. Dazu werden mit Bausteintypen gewisse Eigenschaften wie Funktionalität, Ausführungszeit und Größe eines Hardware-Bausteins verbunden. Um allgemein eine parallele Ausführung der Operationen zu ermöglichen, werden in der Regel mehrere Instanzen eines Bausteintyps zur Verfügung gestellt. Die Zuweisung der Operationen zu den Instanzen dieser Bausteintypen erfolgt durch die *Ressourcen-Bindung*. Da die Zuweisung der Operationen zu Kontrollschritten direkte Auswirkungen auf die anderen Teilaufgaben hat, dürfen die Teilprobleme für optimale Ergebnisse nicht einzeln gelöst werden. Die dazu erforderliche globale Betrachtungsweise von Scheduling, Allokation und Ressourcen-Bindung bedeutet jedoch die Lösung eines NP-harten Problems [GJ79].

In der Mikroarchitektur-Synthese werden als Bewertungskriterien für einen Entwurf einerseits die Kosten[6] und andererseits die Ausführungszeit in Kontrollschritten des Entwurfs herangezogen. Der zwischen Kosten und Zeit bestehende direkte Zusammenhang wird durch den Begriff *Area-Time trade-off* ausgedrückt. So schließt sich in der Regel die Realisierung eines billigen und gleichzeitig schnellen Entwurfs aus. Im allgemeinen werden deswegen zwei Vorgehensweisen unterschieden:

1. Kostenminimierung unter Zeitschranken (*time-constrained synthesis*)

2. Zeitminimierung unter Kostenschranken (*resource-constrained synthesis*)

In Abbildung 1.3 wird dieser Zusammenhang graphisch veranschaulicht.

---

[5] Ein Kontrollschritt stellt einen Zustandsübergang innerhalb des Steuerwerks dar.

[6] Die Kosten sind in der Regel abhängig von den benutzten Hardware-Bausteinen und stellen zum Beispiel den Platzbedarf auf dem Chip oder konkrete Bausteinkosten dar.

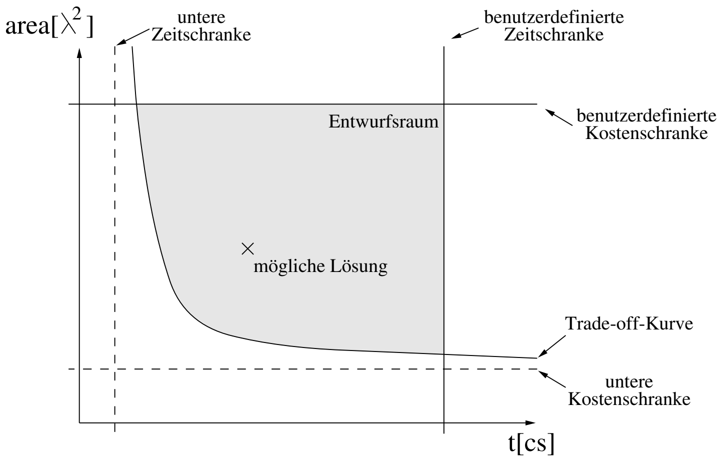


Abbildung 1.3: Area-Time trade-off

Anforderungen an den Chip, zum Beispiel die Einhaltung von Echtzeit-Bedingungen oder geringer Stromverbrauch, führen für jeden Entwurf zu einer Zeit- und Kostenschranke, die absolute untere Grenzen vorgeben (siehe Abbildung 1.3). Optimale Entwürfe liegen auf der *Trade-off-Kurve*. Abweichungen oberhalb von dieser Linie bedeuten automatisch suboptimale Ergebnisse. Neben diesen Schranken kann der Entwurfsraum zusätzlich durch definierte Vorgaben vom Benutzer eingeschränkt werden. So soll bei der Vorgabe einer Kostenschranke der Entwurf mit der geringsten Ausführungszeit bestimmt werden, der die spezifizierten Kosten nicht überschreitet. Im anderen Fall werden bezüglich einer maximal erlaubten Ausführungszeit die minimalen Kosten bestimmt. Die untere Zeitschranke eines Entwurfs wird dabei durch den kritischen Pfad im Datenflußgraphen vorgegeben, der den längsten möglichen Pfad darstellt. Alle Operationen, die auf dem kritischen Pfad liegen, besitzen keine Freiheitsgrade bezüglich der Zuweisung zu Kontrollschritten. Abbildung 1.4 stellt einen Datenflußgraphen mit den sechs einzyklischen Operationen OP_1 bis OP_6 dar.

Die Operationenfolge OP_2, OP_4, OP_5 und OP_6 bildet den kritischen Pfad und erzwingt eine minimale Ausführungszeit von 4 Kontrollschritten. Im Gegensatz zu den auf dem kritischen Pfad liegenden Operationen, können OP_1 und OP_3 innerhalb der ASAP[7]/ALAP[8]-Bereiche Kontrollschritten zugewiesen werden. Die ASAP und ALAP Kontrollschritte einer Operation entsprechen jeweils dem frühest- und spätestmöglichen Ausführungszeitpunkt. Für Operation OP_3 ergibt sich somit als ASAP-Wert Kontrollschritt 1 und als ALAP-Wert Kontrollschritt 2. Falls zur Ausführung der Operationen zwei zusätzliche Kontrollschritte erlaubt werden, führt das zu einer Erhöhung der Freiheitsgrade aller Operationen, da sich die ALAP-Werte um jeweils zwei Kontrollschritte nach hinten verschieben. Zum Beispiel kann die Ausführung von Operation OP_5 dann in den Kontrollschritten 3, 4 oder 5 erfolgen.

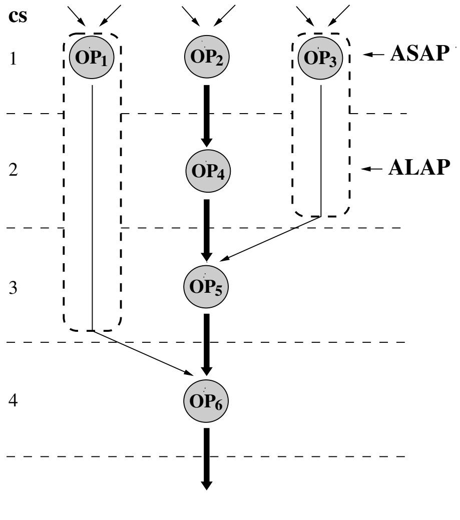


Abbildung 1.4: Kritischer Pfad in einem Datenflußgraphen

[7]ASAP (As Soon As Possible)
[8]ALAP (As Late As Possible)

Die Angabe zusätzlicher Kontrollschritte führt in der Regel zu einer Kostenreduzierung des Designs, da einzelne Bausteine besser ausgelastet werden können. Eine Kostenreduzierung bedeutet jedoch eine Erhöhung der Freiheitsgrade der Operationen und führt zu einer exponentiellen Erhöhung der Rechenzeit (siehe Abbildung 1.5).

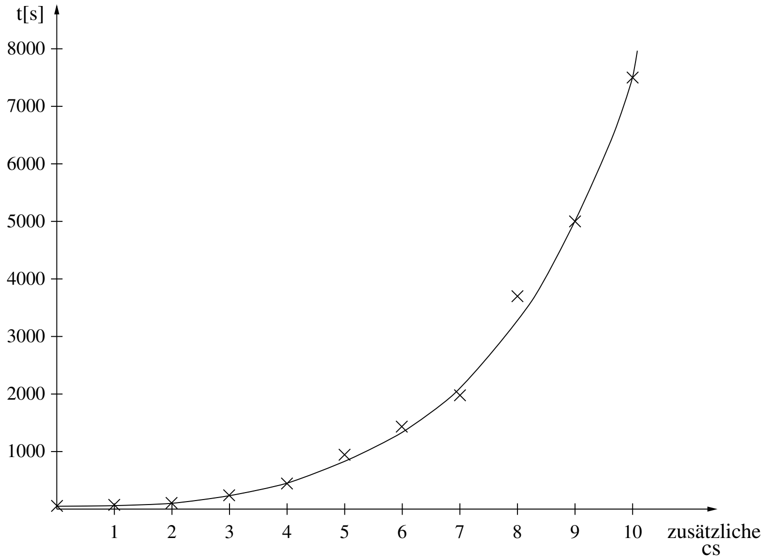


Abbildung 1.5: Komplexität bei Kontrollschritt-Erhöhung

Als Beispiel dient hier der Elliptical-Wave-Filter [KWK85][9].

1.2 Bestehende Verfahren

Zur Durchführung der Mikroarchitektur-Synthese existieren eine Reihe von Verfahren, für die in [Mar93] ein Überblick gegeben wird. Zu den einfachen Scheduling-Verfahren zählen das *ASAP-* und *ALAP-Scheduling*. Bei diesen Verfahren wird eine Zuweisung der Operationen zu Kontrollschritten anhand der frühestmöglichen beziehungsweise spätestmöglichen Ausführungszeitpunkte der entsprechenden Operationen vorgenommen. Beim *List-Scheduling* [Bak74] werden die Operationen, die zum aktuellen Kontrollschritt ausgeführt werden können, bezüglich eines heuristischen Auswahlkriteriums nach Prioritäten sortiert und aufgrund dieser Sortierung Kontrollschritten zugewiesen. Als Prioritätskriterium kann zum Beispiel die Mobilität[10] dienen, indem zuerst die Operationen mit der geringsten Beweglichkeit zugewiesen werden. Eine Variante dieses Verfahrens wird in [KKB91] vorgestellt. Mit dem *kräftebasierten Scheduling* (*Forced-Directed Scheduling*) [PK87] wird eine möglichst gleichmäßige Verteilung der Operationen auf Kontrollschritte angestrebt, indem mögliche Auswirkungen der Zuordnung einer Operation anhand von Wahrscheinlichkeiten berücksichtigt werden. Das in [CT90] beschriebene Verfahren stellt eine Erweiterung des kräftebasierten Scheduling dar und ermöglicht, im Gegensatz zu den meisten (heuristischen) Verfahren, eine globale Betrachtungsweise der Mikroarchitektur-Synthese.

[9] Die Optimierungen wurden mit dem Synthese-System *OSCAR* ([LMD94, Döm94, Lan97]) mittels Methoden der ganzzahlig linearen Programmierung durchgeführt. Die Lösung der linearen Gleichungssysteme erfolgte mit dem MILP-Solver *lp_solve* auf einem Cyrix-Prozessor (Taktfrequenz 133 MHz).

[10] Die Mobilität einer Operation ergibt sich aus der Differenz der ALAP- und ASAP-Kontrollschritte.

Das Kennzeichen der bisher erwähnten Verfahren ist, daß diese schnell und deterministisch eine nicht zwangsläufig optimale Lösung berechnen.

Mit den Verfahren der ganzzahlig linearen Programmierung (Integer Linear Programming, ILP) werden die Randbedingungen in Form linearer Gleichungssysteme dargestellt. Die in den Nebenbedingungen und der Zielfunktion benutzten Variablen dürfen dabei lediglich ganzzahlige Werte annehmen. Eine Variante stellt die binäre lineare Programmierung (0/1-LP) dar, bei der die Variablen nur die Werte 0 oder 1 annehmen dürfen. Unter Einhaltung der gegebenen Nebenbedingungen, die durch lineare (Un-) Gleichungen ausgedrückt werden, erfolgt die Optimierung einer gegebenen linearen Zielfunktion. Bezüglich dieser Zielfunktion und den gegebenen Randbedingungen wird die Berechnung optimaler Ergebnisse sichergestellt. Allerdings bedeutet die Berechnung der ganzzahligen Lösung des Gleichungssystems die Lösung eines NP-harten Problems [PS82]. Der Vorteil dieser Verfahren ist darin zu sehen, daß für kleinere Entwürfe sehr schnell eine optimale Lösung bezüglich der Nebenbedingungen und der gegebenen Zielfunktion berechnet werden kann. Für komplexere Problemstellungen wird die Lösung des Gleichungssystems allerdings viel Zeit in Anspruch nehmen. Die formale Betrachtungsweise ermöglicht eine einfache Erweiterung des Funktionsumfangs durch Ergänzung weiterer Nebenbedingungen. Ein Synthese-System, das auf der Basis der ganzzahlig linearen Programmierung beruht, stellt zum Beispiel das OSCAR-System[11] [LMD94, Döm94, Lan97] dar. Weitere Ansätze werden in [Ach95, GE91] beschrieben, wobei das OSCAR-System gegenüber diesen Synthese-Systemen wesentliche Erweiterungen[12] berücksichtigt.

Die probabilistischen Verfahren, wie genetische Algorithmen (GA), haben sich in der Vergangenheit zur Optimierung komplexer Problemstellungen bewährt. Diese Verfahren arbeiten zwar langsamer als heuristische, sind allerdings aufgrund ihrer Suchoperatoren in der Lage, den Lösungsraum besser zu durchsuchen. Insbesondere werden bei komplexen Entwürfen die Vorteile dieser Verfahren deutlich. Eine Beschreibung der Durchführung der Mikroarchitektur-Synthese mit genetischen Algorithmen erfolgt in [Hei96]. Weitere Verfahren in diesem Bereich werden in [WGH90, ASB94, DHSB95] vorgestellt. Diese führen alle eine gleichzeitige Minimierung der Hardware-Kosten und der benötigten Anzahl von Kontrollschritten durch. In [WGH90] werden zusätzlich mehrzyklische Bausteintypen und funktionales Pipelining unterstützt. Als Erweiterung dazu können mit der in [DHSB95] vorgestellten Methode strukturelles Pipelining und die Behandlung von Conditional-Code und Schleifen berücksichtigt werden. Diese Verfahren ermöglichen zwar eine globale Betrachtung der Teilprobleme, berücksichtigen jedoch bislang nur einen sehr geringen Funktionsumfang. Ein weiteres Verfahren auf der Basis genetischer Algorithmen, das allerdings nur eine Betrachtung von Allokation und Bindung erlaubt, wird in [Man95] vorgestellt. Mit dem in [LM93] beschriebenen Verfahren erfolgt die Durchführung der Mikroarchitektur-Synthese auf der Basis des Simulated Annealing, das jedoch nur eine sequentielle Betrachtung der Teilprobleme ermöglicht.

1.3 Zielsetzung der Arbeit

Da die optimale Lösung der Mikroarchitektur-Synthese die Lösung eines NP-harten Problems bedeutet, kann es keinen Algorithmus geben, der die Berechnung eines global optimalen Ergebnisses bei polynomieller Rechenzeit garantieren kann[13]. Es müssen also Einschränkungen entweder bezüglich der Rechenzeit oder der Optimalität des Ergebnisses vorgenommen werden.

[11] OSCAR (Optimum Simultaneous Scheduling, Allocation and Resource Binding)

[12] Das OSCAR-System unterstützt komplexe Bausteinbibliotheken [MLD96, MLD97], allgemeines Chaining und Verbindungsminimierung. Dabei wird die simultane Betrachtung aller Teilprobleme ermöglicht.

[13] Unter der Voraussetzung, daß $P \neq NP$ gilt.

Mit den Verfahren der ganzzahlig linearen Programmierung besteht die Möglichkeit, die drei Teilprobleme für eine gegebene lineare Zielfunktion global optimal zu lösen. Aufgrund des exponentiellen Verhaltens der Rechenzeit werden allerdings nur bei kleinen Entwürfen Lösungen in vertretbarem zeitlichen Rahmen erreicht. Um auch komplexere Problemstellungen lösen zu können, muß somit die Forderung nach optimalen Ergebnissen abgeschwächt werden. Mit den genetischen Algorithmen stehen effiziente Verfahren zur Verfügung, die zwar keine optimalen Ergebnisse garantieren können, aber häufig gute oder sogar optimale Ergebnisse liefern.

Das Ziel dieser Arbeit besteht in der Entwicklung eines genetischen Algorithmus, der sowohl die drei Hauptaufgaben der Mikroarchitektur-Synthese simultan löst als auch den folgenden Funktionsumfang berücksichtigt:

- Chaining
- Unterstützung von Zeitvorgabevorschriften
- Unterstützung von Komplexbausteinen
- Unterstützung von Bausteinen unterschiedlicher Ausführungsgeschwindigkeiten
- Unterstützung von mehrzyklischen Bausteinen
- Unterstützung von Pipeline-Bausteinen
- Verbindungsminimierung

Es wird vorausgesetzt, daß die ganzzahlig lineare Programmierung die Formulierung des Gesamtproblems durch ein IP-Modell ermöglicht und somit als Grundlage der Optimierung durch den genetischen Algorithmus dient. Mit Hilfe des IP-Modells wird aus der Verhaltensbeschreibung eines Entwurfs eine IP-Datei erzeugt. Diese enthält ein lineares Gleichungssystem mit der zu optimierenden Zielfunktion und den Nebenbedingungen das es zu lösen gilt. Anstelle eines IP-Solver soll ein genetischer Algorithmus zur Optimierung eingesetzt werden. In Abbildung 1.6 wird die Einordnung des genetischen Algorithmus in den Synthese-Ablauf verdeutlicht.

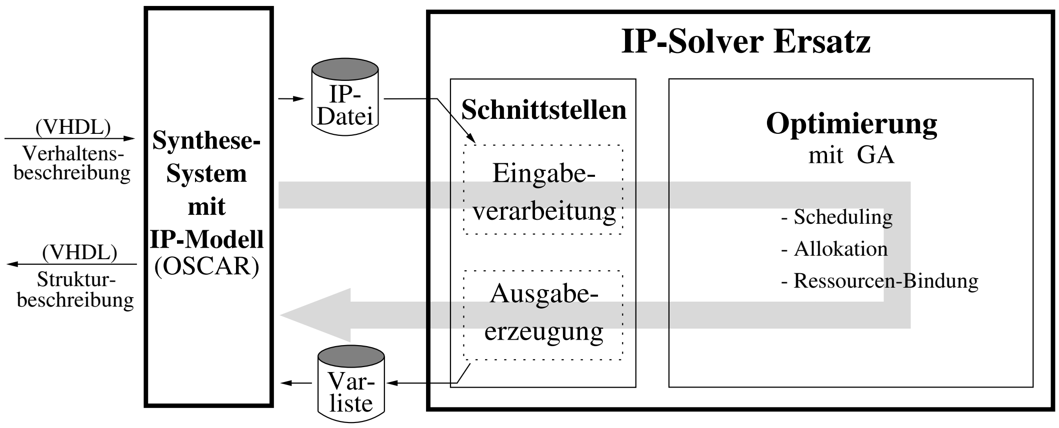


Abbildung 1.6: Synthese-System mit GA als Ersatz für IP-Solver

Anhand der IP-Datei können die zur Optimierung durch den genetischen Algorithmus erforderlichen Informationen gewonnen werden. Die Korrektheit der vom genetischen Algorithmus berechneten besten Lösung wird im Sinne der Correctness-by-Construction durch Einsetzen der Variablen in die IP-Datei nachgewiesen. Die Rückgabe der verifizierten Lösung an das Synthese-System erfolgt schließlich in Form einer Variablenliste.

Das folgende Kapitel gibt eine Einführung in die Arbeitsweise der genetischen Algorithmen. In Kapitel 3 erfolgt dann eine Beschreibung des im Rahmen dieser Diplomarbeit entwickelten genetischen Algorithmus und die Umsetzung der gewonnenen Erkenntnisse zu einem Gesamtsystem in Verbindung mit dem OSCAR-System. Das 4. Kapitel enthält die Darstellung einiger Testergebnisse. Eine Zusammenfassung dieser Arbeit und ein Ausblick wird in Kapitel 5 gegeben.

Kapitel 2

Grundlagen genetischer Algorithmen

In der Natur findet ein ständiger Optimierungsprozeß der Lebewesen aufgrund sich verändernder Umweltbedingungen statt. Eine gute Anpassung der Individuen an vorhandene Umweltbedingungen ist nach dem Prinzip des *survival of the fittest* für das Überleben wichtig. Angepaßte Individuen besitzen gute Überlebenschancen und vererben ihre Gene der nächsten Generation. Im Zuge der Generationen werden dann immer besser an die Umwelt angepaßte Individuen erzeugt.

Die genetischen Algorithmen [Hol92, Gol89] nehmen sich die Natur als Vorbild und lösen Optimierungsprobleme, indem sie den biologischen Evolutionsprozeß nachahmen. Dazu besteht in einem genetischen Algorithmus eine Population aus mehreren Individuen, die für sich genommen jeweils eine potentielle Lösung des Optimierungsproblems darstellen. Durch die Anwendung genetischer Operatoren werden dann im Verlauf der Generationen immer bessere Individuen erzeugt, indem gutes Genmaterial bevorzugt weitervererbt und zufällig verändert wird. In der Regel bleibt die Anzahl der Individuen der Population in jeder Generation konstant[1]. Die Repräsentation eines Individuums erfolgt mittels eines *Chromosoms*, das in einzelne *Gene* unterteilt ist. Durch die Gene werden die Variablen des Optimierungsproblems dargestellt, für die eine optimale Belegung gefunden werden soll. Belegungen der Gene sind konkrete Ausprägungen und werden *Allele* genannt. Falls zum Beispiel eine binäre Kodierung der Gene zugrunde gelegt wird, dürfen als Allele nur die Werte 0 und 1 benutzt werden.

In einem genetischen Algorithmus wird eine Suche nach dem globalen Optimum parallel von mehreren Punkten des Suchraums aus gestartet. Im Gegensatz dazu beginnt die Suche bei herkömmlichen Verfahren nur von einem Punkt aus, wodurch sich in der Regel eine größere Abhängigkeit von der Wahl des Startpunktes ergibt. Die Gefahr einer Konvergenz im Attraktionsbecken eines lokalen Optimums ist dann größer.
Da bei den genetischen Algorithmen keine Voraussagen möglich sind, ob das globale Optimum bereits gefunden wurde oder in den kommenden Generationen weitere Verbesserungen erzielt werden, stehen unterschiedliche Abbruchkriterien zur Auswahl. Zum Beispiel besteht die Möglichkeit den Optimierungsprozeß bei einer zu großen Ähnlichkeit der Individuen, nach einer zuvor bestimmten Anzahl von Generationen oder, falls innerhalb einer gewissen Anzahl von Generationen keine Verbesserung erreicht wurde, zu beenden.

Mittlerweile existieren eine Reihe unterschiedlicher, problemangepaßter genetischer Algorithmen, deren grundsätzlicher Ablauf dem nachfolgend beschriebenen Grundalgorithmus ähnlich ist. Im Anschluß daran erfolgt eine konkretere Beschreibung der einzelnen Unterschritte.

[1] Eine Ausnahme wird zum Beispiel in [LT93] vorgestellt, indem unter anderem die Populationsgröße dynamisch an aktuelle Bedingungen angepaßt wird.

In Abbildung 2.1 ist der allgemeine Ablauf eines genetischen Algorithmus wiedergegeben. Nach der Initialisierung werden alle Individuen der Population bewertet. Auf der Basis der durchgeführten Bewertung wird die Selektion durchgeführt, bei der die Individuen ausgesucht werden, die ihre Gene in die nächste Generation vererben. Zur Erzeugung von Nachkommen wird mit einer Wahrscheinlichkeit von $prob_{cross} \in [0, 1]$ ein Crossover zweier selektierter Eltern durchgeführt. Im folgenden Schritt werden die Gene der Nachkommen vor Übernahme in die nächste Generation mit einer Wahrscheinlichkeit von $prob_{mutat} \in [0, 1]$ einer Mutation unterzogen. Die anschließend durchgeführte Bewertung der Individuen dient wiederum als Grundlage für die Selektion (in Schritt drei), falls die Abbruchbedingung nicht erfüllt ist. Ansonsten wird das bisher beste Individuum ausgegeben.

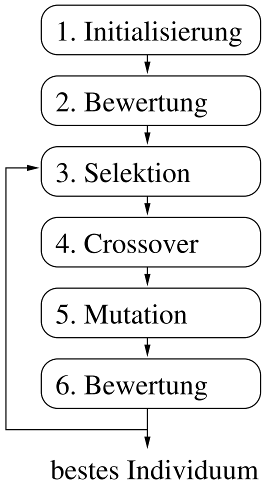


Abbildung 2.1: Ablauf eines GAs

Eine Einführung in diese Verfahren wird unter anderem in [Nis97, Hei94] gegeben.

2.1 Initialisierung

In der Initialisierungsphase wird allen Individuen eine Anfangsbelegung der Gene zugewiesen. Um die Wahrscheinlichkeit zu erhöhen, das globale Optimum im Verlauf der Optimierung zu finden, wird zunächst eine möglichst gute Verteilung der Individuen im Suchraum angestrebt. Belegungen der Gene werden deswegen in der Regel probabilistisch vorgenommen.

2.2 Bewertung

Zur Unterscheidung der Individuen wird mittels einer Bewertungsfunktion jedem Individuum ein bestimmter Wert zugewiesen, der Auskunft darüber gibt, wie gut ein bestimmtes Individuum das Optimum approximiert. Die Bewertungsfunktion wird dabei aus den Zielkriterien des Optimierungsproblems hergeleitet und muß eine schnelle Auswertung der Individuen erlauben, da im Verlauf der Generationen eine große Anzahl Berechnungen erforderlich sind. Besteht die Aufgabe zum Beispiel in der Minimierung einer Kostenfunktion, so ist es denkbar, daß diese Kostenfunktion gleichzeitig als Bewertungsfunktion dient. Erfolgt die Kostenminimierung unter Nebenbedingungen, so können allerdings noch zusätzliche Erweiterungen erforderlich sein (siehe Abschnitt 2.6). Durch die Bewertungsfunktion zugewiesene Werte sind immer nur vom jeweiligen Individuum abhängig. Um eine Relation der Güte eines einzelnen Individuums zu der der Gesamtbevölkerung zu schaffen, werden in der Regel diese Werte in *Fitneßwerte* transformiert. So kann es trotz einer schlechten Bewertung vorkommen, daß ein Individuum eine hohe Fitneß zugewiesen bekommt, wenn der Rest der Population noch schlechter bewertet wurde.

2.3 Selektion

Mit der Durchführung der Selektion werden diejenigen Individuen ausgewählt, die ihre Gene in die nächste Generation vererben sollen. Die Auswahl erfolgt dabei anhand der Fitneß eines

Individuums, wobei eine hohe Fitneß mit einer großen Selektionswahrscheinlichkeit verbunden ist.

Ein Selektionsverfahren, bei dem die Selektionswahrscheinlichkeit eines Individuums *ind* vom Verhältnis der eigenen Fitneß zur Summe der Fitneßwerte der Gesamtpopulation abhängig ist, stellt die in Abbildung 2.2 dargestellte *fitneßproportionale Selektion* dar. Veranschaulicht wird diese Vorgehensweise anhand eines Glücksrads, das in soviele Felder unterteilt ist, wie Individuen in der Population vorhanden sind. Die Größe der Abschnitte entspricht dabei dem anteiligen Fitneßwert des dazugehörigen Individuums zur Gesamtpopulation. Das Glücksrad wird nun so oft gedreht, wie Individuen in den Genpool kopiert werden sollen. Das Individuum, auf das der Zeiger in jedem Durchgang weist (hier ind_3), wird in den Genpool übernommen.

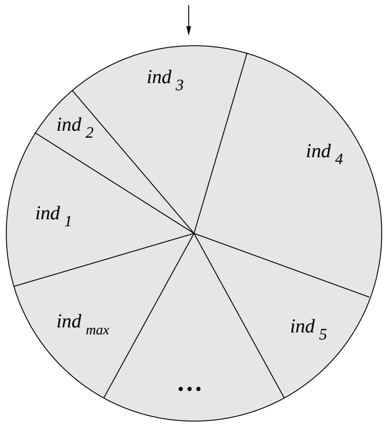


Abbildung 2.2: Fitneßproportionale Selektion

Das Stochastic Universal Sampling (SUS) stellt eine Erweiterung der fitneßproportionalen Selektion dar. Bei dieser Variante sind soviele Zeiger in gleichmäßigen Abständen am Rand der Scheibe positioniert, wie Individuen in den Genpool übernommen werden sollen. Nach einmaligem Drehen des Glücksrads, werden dann soviele Kopien von einem Individuum erzeugt, wie Zeiger auf den entsprechenden Abschnitt verweisen.

2.4 Crossover

Das Crossover stellt den zentralen Suchoperator in einem genetischen Algorithmus dar und erfüllt die Aufgabe, aus dem vorhandenen Genmaterial der Population neue Individuen zu erzeugen. Nach einer zufälligen Auswahl zweier Eltern aus dem Genpool werden die Gene derart kombiniert, daß zwei Nachkommen entstehen, die teilweise aus Genen des einen und des anderen Elter bestehen.

Die einfachste Möglichkeit zur Durchführung eines Crossover stellt das in Abbildung 2.3 dargestellte *Ein-Punkt Crossover* dar. In diesem Beispiel besteht ein Chromosom aus jeweils 6 Genen, die mit binären Allelen belegt sind. Zur Durchführung des Ein-Punkt Crossover wird zufällig eine Stelle auf dem Chromosom bestimmt, an der die Chromosomen der ausgewählten Eltern aufgespalten werden. In diesem Beispiel findet ein Crossover von Elter 1 und Elter 2 zwischen den Genen 2 und 3 statt. Es werden dabei die Gene ausgetauscht, die hinter dieser Stelle auf dem Chromosom liegen. Der Austausch der Gene 3 bis 6 führt dann zu den Nachkommen 1 und 2.

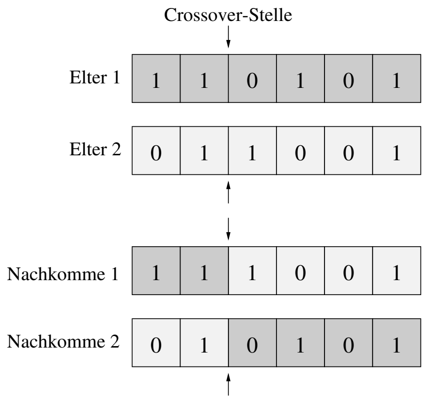


Abbildung 2.3: Ein-Punkt Crossover

Analog dazu erfolgt die Durchführung des *Zwei-Punkt Crossover*, bei dem die Gene zweier Eltern zwischen zwei zufällig bestimmten Stellen des Chromosoms ausgetauscht werden.
Beim *Uniform Crossover* wird hingegen für jedes Gen einzeln entschieden, ob es ausgetauscht werden soll. Während beim Ein-Punkt Crossover die Austauschwahrscheinlichkeit von Genen,

die weiter vorne auf dem Chromosom liegen geringer ist, als bei weiter hinten liegenden, zeichnet sich das Uniform Crossover durch eine identische Austauschwahrscheinlichkeit für alle Gene aus.

2.5 Mutation

Da in einem genetischen Algorithmus das Genmaterial guter Individuen bevorzugt weitervererbt wird, besteht die Gefahr, daß sich ein großer Teil der Population vorzeitig einem lokalen Optimum nähert. Dabei werden qualitativ schlechte Individuen verdrängt, die durchaus Allele besitzen können, die für ein besseres Optimum notwendig sind. Die Mutation erfüllt deswegen die Aufgabe, neues Genmaterial zu erzeugen oder im Verlauf des Evolutionsprozesses verlorengegangenes wiederzubeschaffen. Im Gegensatz zur Durchführung des Crossover wird die Mutation auf einzelne Individuen der Population angewendet, indem Genbelegungen zufällig verändert werden.

In Abbildung 2.4 wird die Mutation am Beispiel des zuvor erzeugten Nachkommen 1 verdeutlicht, bei dem das Allel von Gen 2 verändert werden soll. Im Fall der in diesem Beispiel vorliegenden binären Kodierung der Gene führt die Mutation von Gen 2 zu einem Wechsel des Allels von 1 auf 0.

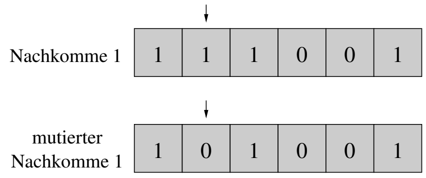


Abbildung 2.4: Mutation eines Gen

2.6 Einbeziehung von Nebenbedingungen

In vielen Optimierungsproblemen, wie auch bei der Durchführung der Mikroarchitektur-Synthese, müssen für gültige Lösungen eine Reihe von Nebenbedingungen eingehalten werden, die den Lösungsraum einschränken. Das können zum Beispiel Vorgaben bestimmter Grenzwerte für Variablen sein, die weder über- noch unterschritten werden dürfen, oder Beziehungen zwischen den Variablen. Die Berücksichtigung von Nebenbedingungen in einem genetischen Algorithmus kann grundsätzlich auf die folgenden Arten erfolgen:

- Jeder Nebenbedingung wird ein Strafwert zugeordnet, der die Qualität eines Individuums bei Verstoß gegen diese Nebenbedingung verringert. Besteht die Aufgabe in der Minimierung einer Kostenfunktion unter Berücksichtung von Nebenbedingungen, so kann die Kostenfunktion in Verbindung mit einem zusätzlichen Strafterm als Bewertungsfunktion dienen. Der Strafterm führt dann für jede nicht eingehaltene Nebenbedingung zu einer Erhöhung der bereits angefallenen Kosten, wodurch die Güte des bestraften Individuums verringert wird.
- Mit der Verwendung einer speziellen Kodierung der Individuen wird die Wahrscheinlichkeit von Verstößen gegen Nebenbedingungen verringert. Falls dennoch ungültige Lösungen generiert werden, erfolgt eine Korrektur mittels eines Repair-Algorithmus.
- Die Einführung spezieller genetischer Operatoren, die die Nebenbedingungen in sich enthalten, führt dazu, daß die Gültigkeit der Individuen nicht verloren geht.

Es besteht weiterhin die Möglichkeit, die hier vorgestellten Integrationsmöglichkeiten von Nebenbedingungen in einem genetischen Algorithmus miteinander zu kombinieren.

Kapitel 3

Mikroarchitektur-Synthese mit genetischen Algorithmen

Die hier vorgestellte Durchführung der Mikroarchitektur-Synthese mit genetischen Algorithmen beginnt mit dem Einlesen der von einem Synthese-System generierten IP-Datei. In dieser IP-Datei sind alle zur Realisierung der Synthese erforderlichen Informationen in Form einer linearen Zielfunktion und linearer Nebenbedingungen enthalten. Die Aufgabe des genetischen Algorithmus besteht nun in der Minimierung der angegebenen Zielfunktion unter Einhaltung der Nebenbedingungen. Das Ziel ist die Berechnung einer guten, eventuell global optimalen Lösung, in polynomieller Zeit.

Im folgenden Abschnitt werden zunächst die grundlegenden mathematischen Notationen vorgestellt. Die Einführung weiterer Notationen erfolgt bei Bedarf im Verlauf dieses Kapitels. Anschließend wird die grundsätzliche Vorgehensweise des realisierten genetischen Algorithmus erläutert. Die Einhaltung der Vorschriften durch den genetischen Algorithmus stellt das Hauptroblem bei der Mikroarchitektur-Synthese dar und bedarf deswegen einer detaillierten Beschreibung. Die darauffolgenden Abschnitte beschäftigen sich mit der zu minimierenden Zielfunktion und den zu berücksichtigenden Synthesevorschriften. Zum Abschluß dieses Kapitels wird die konkrete Realisierung des genetischen Algorithmus und dessen Integration in ein Gesamtsystem vorgestellt.

3.1 Notationen und Begriffsbestimmungen

In diesem Abschnitt werden die grundlegenden Notationen zur Durchführung der Mikroarchitektur-Synthese eingeführt[1]. Dieses sind insbesondere Notationen für Operationen, Kontrollschritte, Bausteintypen und Instanzen.
Im Verlauf der Synthese muß jede *Operation* j einem *Kontrollschritt* $i \in I$, einem *Bausteintyp* $m \in M$ und einer *Instanz* $k \in K$ eines Bausteintyps zugewiesen werden. I, M und K stellen also Mengen dar, aus denen jeweils ein Element für jede Operation $j \in J$ ausgewählt wird. Im einzelnen gilt:

Menge der Operationen $J \subset I\!N$, $j \in J = \{1, \ldots, j_{max}\}$

Menge der Kontrollschritte $I \subset I\!N$, $i \in I = \{1, \ldots, i_{max}\}$

Menge der Bausteintypen $M \subset I\!N$, $m \in M = \{1, \ldots, m_{max}\}$

[1] Diese Notationen orientieren sich an [Döm94, LMD94].

Menge der Instanzen $K \subset I\!N$, $k \in K = \{1, \ldots, k_{max}\}$

In der Regel werden mehrere Instanzen k eines Bausteintyps m zur Verfügung gestellt, deren Zuordnung zu einem Bausteintyp im folgenden mit $type(k)$ ausgedrückt wird. Zur Durchführung der Synthese sind allerdings noch weitere Angaben über Eigenschaften von Instanzen bestimmter Bausteintypen erforderlich: Die Kenntnis der Ausführungsdauer einer bestimmten Operation j auf der Instanz k ist zur Einhaltung von Datenabhängigkeiten zwischen den einzelnen Operationen notwendig und wird durch $C(j,k)$ ausgedrückt. Gilt zum Beispiel $C(j,k) = 2$ cs, so liegt das Ergebnis von Operation j bei Ausführung auf der Instanz k nach 2 Kontrollschritten vor. Eine bestehende *Datenabhängigkeit* zwischen zwei Operationen j und j' wird symbolisch durch $j \prec j'$ ausgedrückt. In diesem Fall benötigt die Operation j' das Ergebnis von Operation j zur Ausführung auf einer Instanz. Falls die Operation j in Kontrollschritt i auf der Instanz k ausgeführt wird, darf also die Operation j' frühestens zu Kontrollschritt $i' = i + C(j,k)$ gestartet werden. Die Ausführung von Operation j auf der Instanz k führt ebenso dazu, daß die benutzte Instanz eine bestimmte Anzahl von Kontrollschritten nicht mit einer neuen Operation belegt werden darf. Dieses wird durch die *Latenzzeit* $\ell(j,k)$ beschrieben.

Eine Übersicht aller verwendeten Notationen ist im Anhang B in tabellarischer Form angegeben.

3.2 Vorüberlegungen zur Arbeitsweise

In diesem Abschnitt werden Überlegungen zur Wahl einer geeigneten Kodierung der Individuen und zur grundsätzlichen Vorgehensweise, mit der die Nebenbedingungen berücksichtigt werden können, dargelegt. Eine detaillierte Beschreibung der Einhaltung der Nebenbedingungen erfolgt in Abschnitt 3.4.

Für die Durchführung der Mikroarchitektur-Synthese mit einem genetischen Algorithmus ist die Entwicklung einer Methode erforderlich, mit der die generierten Nebenbedingungen berücksichtigt werden können. Jedes Individuum der Population soll dabei eine potentielle Lösung repräsentieren. Aufgrund der Komplexität des Problems ist zu beachten, daß eventuell nicht immer alle Nebenbedingungen von einem Individuum eingehalten werden können. Für den Ablauf des genetischen Algorithmus ist es deswegen erforderlich, Lösungen, die nicht den spezifizierten Anforderungen genügen, zu erkennen und gegenüber gültigen Lösungen entsprechend abzuwerten.

Die Minimierung der Zielfunktion soll dabei unter Einhaltung einer Zeitschranke geschehen, die bei Überschreiten automatisch zu ungültigen Lösungen führt. Für jeden Entwurf ist es möglich, eine zusätzliche Anzahl von Kontrollschritten anzugeben. Die sich ergebenden ASAP/ALAP-Bereiche müssen bei der Zuweisung der Operationen zu einem Kontrollschritt eingehalten werden, um eine Ausführung des Entwurfs innerhalb der festgelegten Zeitschranke zu gewährleisten. Bei der Erzeugung der IP-Datei werden deswegen keine Variablen generiert, die zu erkennbar suboptimalen oder ungültigen Lösungen führen. Dadurch ergeben sich für die einzelnen Operationen bereits erste Zuordnungsrestriktionen in der Art, daß eine Ausführung zu bestimmten Kontrollschritten oder auf bestimmten Bausteintypen aufgrund der Funktionalität und der Ausführungsdauer von vornherein ausgeschlossen wird. So werden Kontrollschritt-Einschränkungen für die Operationen durch Eingrenzung der jeweiligen ASAP/ALAP-Bereiche durchgeführt. Eine Zuweisung von Operation j zu einem Kontrollschritt, der außerhalb dieses Bereiches liegt, führt dann offensichtlich zu einer ungültigen Lösung. Weiterhin bewirkt die Zuweisung einer Operation zu einem Kontrollschritt und einem Hardware-Baustein in der Regel weitere Kontrollschritt-Einschränkungen für andere Operationen. Für eine Operation j werden deswegen alle während der Optimierung durch den genetischen Algorithmus vorgenommenen Eingrenzungen der Kontrollschritte durch *dynamische* Kontrollschritt-Grenzen $\text{ASAP}_{dyn}(j)$ und $\text{ALAP}_{dyn}(j)$ ausgedrückt. Die Zuweisung von Operation j zu einem Kontrollschritt wird dann bezüglich dieser dynamischen Grenzen vorgenommen.

In Abbildung 3.1 ist ein Datenflußgraph mit den Operationen j und j' gegeben, zwischen denen eine Vorrangsvorschrift $j \prec j'$ besteht. Die Zuweisung von Operation j zu Kontrollschritt 2 führt dazu, daß die Ausführung von Operation j' frühestens in Kontrollschritt 3 gestartet werden darf. Berücksichtigt wird diese Einschränkung durch Verändern der $\text{ASAP}_{dyn}(j')$-Grenze von 2 auf 3. Um eine gültige Lösung zu erreichen, muß nun die Operation j' einem Kontrollschritt aus $\{\text{ASAP}_{dyn}(j'), \ldots, \text{ALAP}_{dyn}(j')\} = \{3, 4\}$ zugewiesen werden.

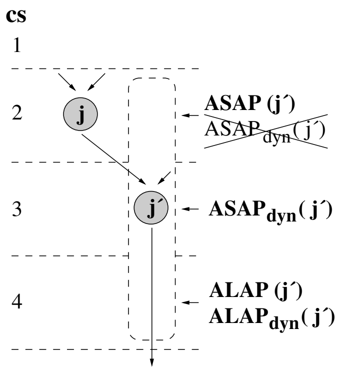


Abbildung 3.1: (Dynamische) ASAP- und ALAP-Grenzen

Ein wichtiger Aspekt bei der Entwicklung von genetischen Algorithmen spielt die chromosomale Darstellung der Individuen, die wesentlich für gute Ergebnisse ist.

Kodierung

Die Kodierung der Individuen wird so gewählt, daß jedes Individuum, dargestellt durch ein Chromosom, eine potentielle Lösung repräsentiert. Da jede Operation des Datenflußgraphen genau einem Kontrollschritt und einem Hardware-Baustein zugeordnet werden muß, bietet sich dazu eine Kodierung mit zwei Genen für jede Operation an. Die Belegungen der Gene ermöglichen dann eine Interpretation als Kontrollschritt und Hardware-Baustein einer bestimmten Operation. In Abbildung 3.2 ist die gewählte Kodierung dargestellt.

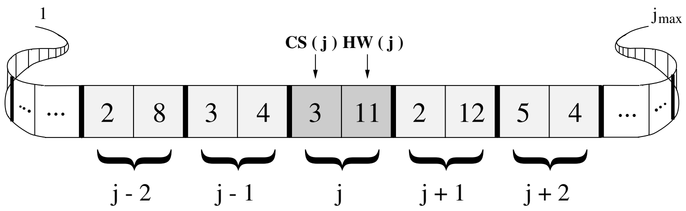


Abbildung 3.2: Kodierung eines Individuums

Zuweisungen werden durch zwei Gene repräsentiert, indem das erste Gen von Operation j die Zuordnung zu einem Kontrollschritt $CS(j)$, das zweite die Zuordnung zu einem Hardware-Baustein $HW(j)$ angibt. Die Belegung von Kontrollschritt-Gen $CS(j)$ und Hardware-Gen $HW(j)$ in Abbildung 3.2 mit den Werten 3 und 11 bewirkt, daß die Ausführung von Operation j in Kontrollschritt 3 auf Instanz 11 gestartet wird.
O.B.d.A. beginnt die Numerierung der Gene auf dem Chromosom mit der Position 0. Die Gene für Operation j liegen dann an den Positionen $2 * (j-1)$ und $2 * (j-1) + 1$. Als Gesamtlänge eines Chromosoms ergibt sich $2 * j_{max}$, mit j_{max} als Anzahl der Operationen. Es wird vorausgesetzt, daß eine topologische Ordnung der Operationen bezüglich ihrer Ausführungsreihenfolge

vorliegt. So gilt für zwei Operationen j und j', zwischen denen eine Vorrangsvorschrift $j \prec j'$ besteht, daß die Ordnungsnummer von j stets kleiner als die von j' ist. Für die Darstellung der Operationen auf dem Chromosom bedeutet dies, daß Operation j' im Chromosom auf jeden Fall hinter Operation j steht. Durch die topologische Ordnung der Operationen wird eine effiziente Berücksichtigung aller Nebenbedingungen während eines Chromsomendurchlaufs unterstützt.

Das Prinzip des Chromosomendurchlaufs

Die Durchführung eines Chromosomendurchlaufs bedeutet, daß die auf dem Chromosom abgelegten Gene der Operationen von links nach rechts besucht werden. Durch diese Vorgehensweise erfolgt eine Bearbeitung der Operationen von der kleinsten bis zur größten Ordnungsnummer. Während eines Chromosomendurchlaufs werden dann bestehende Beziehungen zu Operationen berücksichtigt, die noch nicht besucht wurden. Aufgrund der topologischen Ordnung kann das nur die Operationen betreffen, die eine höhere Ordnungsnummer als die aktuell besuchte besitzen. Die Vorrangsvorschrift, das Chaining und die Zeitvorgabevorschrift stellen Vorschriften dar, die direkte Auswirkungen auf die dynamischen Kontrollschritt-Bereiche anderer Operationen haben (siehe Abschnitte 3.4.3, 3.4.4 und 3.4.6). Die Baustein-Zuordnungsvorschrift, Makrooperationen und die Verbindungsminimierung (siehe Abschnitte 3.4.2, 3.4.5 und 3.4.7) haben dagegen eher Einfluß auf die Hardware-Zuordnung.

Für die während eines Chromosomendurchlaufs aktuell zu behandelnde Operation j werden die folgenden grundlegenden Arbeitsschritte durchgeführt:

1. Einschränkungen der dynamischen Kontrollschritt-Grenzen von Operation j. Diese Einschränkungen können aufgrund von vorhandenen Zeitvorgabevorschriften zu anderen Operationen vorgenommen werden.

2. Auswahl eines Kontrollschrittes aus $\{\mathrm{ASAP}_{dyn}(j), \ldots, \mathrm{ALAP}_{dyn}(j)\}$ und Belegung des Kontrollschritt-Gens $CS(j)$.

3. Auswahl eines geeigneten Bausteintyps in Form einer Instanz unter Berücksichtigung des zuvor ausgewählten Kontrollschrittes[2] und Belegung des Hardware-Gens $HW(j)$.

4. Einschränkungen der dynamischen Kontrollschritt-Bereiche von Operationen, zu denen aufgrund der Vorschriften eine direkte Beziehung besteht. Dazu gehören Zeitvorgabevorschriften, Vorrangsvorschriften und die Berücksichtigung von Chaining.

Zur Verdeutlichung dieser Vorgehensweise sollen im folgenden Beispiel zwei Operationen j und j' betrachtet werden, zwischen denen eine Vorrangsvorschrift $j \prec j'$ und eine konstante Zeitvorgabevorschrift von 2 Kontrollschritten besteht. Dazu werden die Variablen $min_{timing}(j, j')$ und $max_{timing}(j, j')$ zur Einhaltung des minimalen und maximalen Zeitabstandes der Operationen j und j' auf den Wert 2 cs gesetzt. Die Veränderungen der Kontrollschritt-Bereiche der beiden Operationen zu unterschiedlichen Stadien während des Chromosomendurchlaufs sind in Abbildung 3.3 dargestellt und beziehen sich immer auf die dynamischen ASAP- und ALAP-Werte der Operationen. Änderungen der dynamischen Grenzen werden durch die heller markierten Bereiche gekennzeichnet.

[2] Die Auswahl eines Bausteintypen muß vom Ausführungszeitpunkt abhängig gemacht werden, damit sichergestellt werden kann, daß die Operation auf einem genügend schnellen Bausteintyp ausgeführt wird.

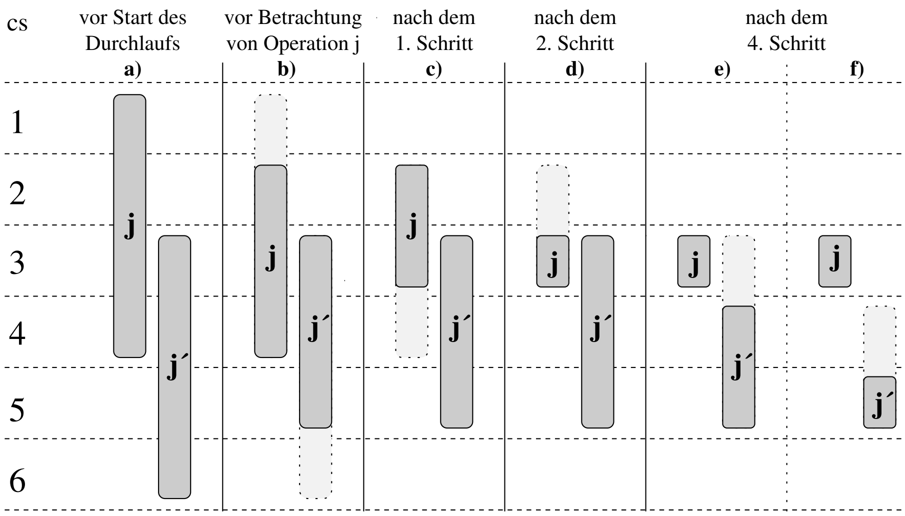


Abbildung 3.3: Veränderung der Kontrollschritt-Bereiche beim Chromosomendurchlauf

In Abbildung 3.3 Spalte a sind die Kontrollschritt-Bereiche der Operationen j und j' vor dem Start des Chromosomendurchlaufs angegeben. Die dynamischen ASAP- und ALAP-Werte stimmen zu diesem Zeitpunkt noch mit den ursprünglichen Werten überein.

Für die Operationen j und j' gilt zu Beginn:

- $\mathrm{ASAP}(j) = \mathrm{ASAP}_{dyn}(j) = 1$ und $\mathrm{ALAP}(j) = \mathrm{ALAP}_{dyn}(j) = 4$
- $\mathrm{ASAP}(j') = \mathrm{ASAP}_{dyn}(j') = 3$ und $\mathrm{ALAP}(j') = \mathrm{ALAP}_{dyn}(j') = 6$

Bevor im Chromosom die Operation j erreicht wird, können die Kontrollschritt-Bereiche der beiden hier betrachteten Operationen j und j' durch vorhandene Beziehungen zu anderen Operationen verändert werden (siehe Spalte b). So wird für Operation j zum Beispiel der frühestmögliche Ausführungszeitpunkt $\mathrm{ASAP}_{dyn}(j)$ von Kontrollschritt 1 auf 2 hochgesetzt und der spätestmögliche Ausführungszeitpunkt $\mathrm{ALAP}_{dyn}(j')$ von Operation j' von Kontrollschritt 6 auf 5 runtergesetzt.

Im ersten Arbeitsschritt werden nun Einschränkungen der *aktuell* zu behandelnden Operation, in diesem Fall also Operation j, vorgenommen. Aufgrund des minimalen Zeitabstandes von 2 Kontrollschritten, wird $\mathrm{ALAP}_{dyn}(j)$ von 4 auf 3 heruntergesetzt (siehe Spalte c). Diese Einschränkung ist sinnvoll, da ein Beginn der Ausführung von Operation j zum Zeitpunkt 4 zu einer ungültigen Lösung führen würde, da Operation j' aufgrund von $\mathrm{ALAP}_{dyn}(j') = 5$ spätestens zu Kontrollschritt 5 gestartet werden muß.

Anschließend wird der Operation j im zweiten Arbeitsschritt probabilistisch ein Kontrollschritt aus $\{\mathrm{ASAP}_{dyn}(j), \ldots, \mathrm{ALAP}_{dyn}(j)\}$ also aus $\{2, 3\}$ zugewiesen. In diesem Fall wurde Kontrollschritt 3 ausgewählt (siehe Spalte d).

Auf der Basis des ausgewählten Kontrollschrittes wird im dritten Arbeitsschritt ein geeigneter Bausteintyp bestimmt und in Form einer Instanz als Allel des Hardware-Gens abgelegt. Hierbei dürfen nur Bausteintypen berücksichtigt werden, auf denen eine genügend schnelle Ausführung möglich ist, um die vorgegebene Ausführungszeit der Schaltung einzuhalten. In Abbildung 3.4 wird dies anhand der beiden Operationen j und j' verdeutlicht.

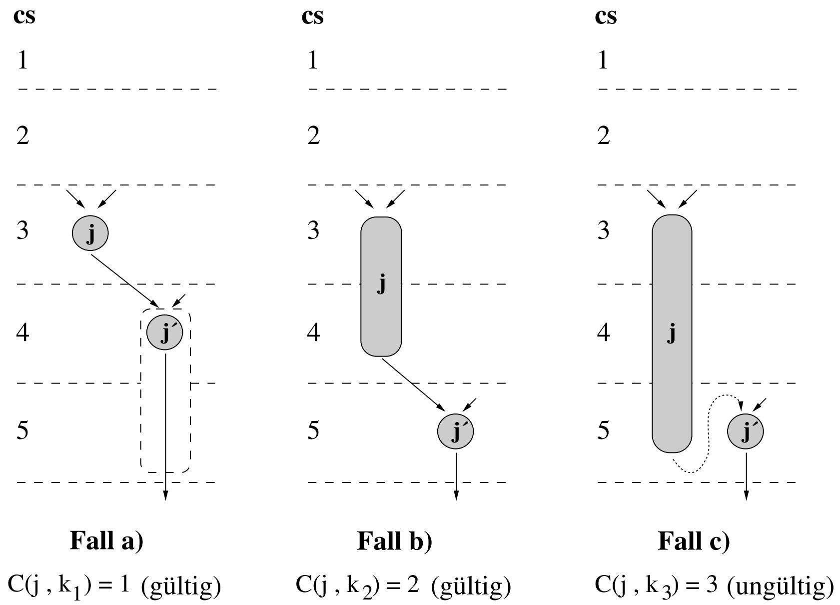


Abbildung 3.4: Auswahl eines hinreichend schnellen Bausteintyps

Als Auswahlmöglichkeiten stehen für die Operation j die Bausteintypen $m_1 = type(k_1)$, $m_2 = type(k_2)$ und $m_3 = type(k_3)$ mit den unter den Datenflußgraphen angegebenen Ausführungszeiten zur Verfügung. Da Operation j im vorherigen Arbeitsschritt bereits Kontrollschritt 3 zugewiesen worden ist, darf aufgrund der bestehenden Vorrangsvorschrift eine Ausführung von Operation j' nur in den Kontrollschritten 4 oder 5 gestartet werden. Während in den beiden ersten Fällen a und b noch eine korrekte Ausführung von Operation j' möglich ist, kann mit der Ausführung von Operation j' in Fall c nicht rechtzeitig begonnen werden. Um gültige Lösungen zu erhalten darf die Operation j also nur auf Instanzen der Bausteintypen m_1 oder m_2 ausgeführt werden. In diesem Beispiel wird Operation j der Instanz k_1 von Bausteintyp m_1 zugewiesen.

Nachdem die Operation j einem Kontrollschritt und einer Instanz zugewiesen wurde, werden im vierten Schritt notwendige Kontrollschritt-Einschränkungen zu Operationen durchgeführt, die beim Chromosomendurchlauf noch nicht besucht wurden, hier also Operation j'. Die aufgrund der vorhandenen Vorrangsvorschrift $j \prec j'$ erforderliche Kontrollschritt-Einschränkung von Operation j' ist in Abbildung 3.3 Spalte e dargestellt.
Die Berücksichtigung des minimalen Zeitabstandes von 2 Kontrollschritten führt zu einer weiteren Einschränkung von Operation j', wie in Spalte f dargestellt. Die Ausführung von Operation j' kann jetzt nur noch in Kontrollschritt 5 gestartet werden.

Anhand dieses Beispiels wurde gezeigt, wie die Freiheitsgrade der Operationen eingeschränkt werden können. Die Auswahl eines Kontrollschrittes für eine Operation j aus $\{\mathrm{ASAP}_{dyn}(j), \ldots, \mathrm{ALAP}_{dyn}(j)\}$ führt stets zu einer gültigen, aber nicht zwangsläufig auch zu einer optimalen Lösung. Durch die Verwendung eines genetischen Algorithmus, der mit einer Population von Individuen arbeitet, kann trotz einzelner suboptimaler Lösungen das globale Optimum gefunden werden, da sich ungünstige Variablenbelegungen nur auf einzelne Individuen, nicht aber zwangsläufig auf die gesamte Population auswirken. Die Bewertung eines Individuums erfolgt nach der Beendung eines Chromosomendurchlaufs. Die Grundlage dazu bildet die in der IP-Datei angegebene Zielfunktion, die im folgenden Abschnitt erklärt wird.

3.3 Zielfunktion

Die in der IP-Datei angegebene Zielfunktion stellt eine lineare Kostenfunktion dar, die minimiert werden soll. Es werden zwei unterschiedliche Kostenquellen berücksichtigt:

1. Die *Instanzenkosten* entsprechen der Summe der Einzelkosten c_k der allozierten Instanzen k.

2. Die *Verbindungskosten* fallen an, wenn Daten zwischen Instanzen ausgetauscht werden müssen. Für eine Verbindung zwischen den Instanzen k und k' entstehen mit $c_{k,k'}$ gewichtete Kosten. Falls vom Designer die Durchführung einer Verbindungsminimierung gewünscht wird, stellen die Verbindungskosten eine Erweiterung der Kostenfunktion dar (siehe auch Abschnitt 3.4.7).

Es ergibt sich als Kostenfunktion:

$$\underbrace{\sum_{k \in K} c_k * b_k}_{Instanzenkosten} + \underbrace{\sum_{k,k' \in K} c_{k,k'} * w_{k,k'}}_{Verbindungskosten} \rightarrow min$$

Die Belegungen der binären Entscheidungsvariablen b_k und $w_{k,k'}$ geben den Ausschlag, ob die mit einer Instanz oder Verdrahtung verbundenen Kosten in die Gesamtkosten eingehen.

Im folgenden Abschnitt wird beschrieben, wie die vom Anwender spezifizierten Eigenschaften des Systems eingehalten werden können. Danach wird in Abschnitt 3.5 die Umsetzung dieser Erkenntnisse in einen genetischen Algorithmus dargelegt.

3.4 Einhaltung der Vorschriften

Bei der Synthese müssen unterschiedliche Vorschriften beachtet werden. Einige dieser Vorschriften sind für einen korrekten Entwurf unbedingt erforderlich und stellen damit die Basis eines gültigen Entwurfs dar. Weitere Spezifikationen können optional vom Anwender angegeben werden. Im folgenden werden zunächst die Methoden beschrieben, mit denen die Basisvorschriften eingehalten werden können. Diese werden durch die Operations-Zuordnungs-, Baustein- und Vorrangsvorschriften repräsentiert. Danach wird auf die Berücksichtigung optionaler Vorschriften eingegangen. Im einzelnen sind dies Chaining, Unterstützung komplexer Bausteine, Zeitvorgabevorschriften und die Verbindungsminimierung.

3.4.1 Operations-Zuordnungsvorschrift

Die Operations-Zuordnungsvorschrift stellt sicher, daß jede in Hardware realisierte Operation j genau einem Kontrollschritt i und genau einer geeigneten Instanz k zugeordnet wird. Voraussetzung dafür ist, daß für jede Operation mindestens ein geeigneter Bausteintyp existiert, auf dem diese ausgeführt werden kann. Die Einhaltung der Operations-Zuordnungsvorschrift kann für jede Operation aufgrund der gewählten Kodierung stets garantiert werden, da jede Operation j durch das Kontrollschritt-Gen $CS(j)$ und das Hardware-Gen $HW(j)$ genau einmal auf einem Chromsom vertreten ist. Dadurch ist eine genaue Zuordnung der Operationen zu einem Ausführungszeitpunkt und einem Hardware-Baustein möglich.

3.4.2 Baustein-Zuordnungsvorschrift

Die Baustein-Zuordnungsvorschrift stellt sicher, daß jeder Instanz k zu jedem Zeitpunkt i maximal eine Operation j zugewiesen wird und daß sie frühestens nach Einhaltung einer Latenzzeit von $\ell(j,k)$ Kontrollschritten neu belegt werden darf. Jede Instanz k wird also mit höchstens einer Operation j in $\ell(j,k)$ Kontrollschritten belegt. Die Einführung von Latenzzeiten für die Bausteintypen ist zur Berücksichtigung von Pipeline-Bausteinen erforderlich. So gilt für diese Bausteine üblicherweise, daß die Gesamtausführungszeit $C(j,k)$ größer als die Latenzzeit $\ell(j,k)$ des Bausteins ist. Es kann also mehr als eine Operation gleichzeitig auf einem solchen Pipeline-Baustein ausgeführt werden, da bereits vor Beendung der Ausführung einer Operation mit der Bearbeitung der nächsten Operation begonnen werden kann.

Die Einhaltung der Baustein-Zuordnungsvorschrift ist möglich, falls für eine Instanz k bei der Ausführung von Operation j sichergestellt werden kann, daß diese während der Latenzzeit $\ell(j,k)$ nicht mit einer neuen Operation belegt wird. Alle erforderlichen Informationen werden deswegen in der Tabelle tab_{bind}[3] gespeichert. Hierbei werden für jeden Kontrollschritt, in dem eine Instanz mit keiner weiteren Operation belegt werden darf, Eintragungen vorgenommen. Unter der Voraussetzung, daß in einem Preprocessing-Schritt von vornherein eine ausreichende Anzahl von Instanzen zur Verfügung gestellt wird, können Verstöße gegen die Baustein-Zuordnungsvorschrift stets verhindert werden. Dazu ist es erforderlich, daß zuerst eine Allokation und zum Schluß die Ressourcen-Bindung durchgeführt wird.

Im Gesamtablauf des genetischen Algorithmus kann die Einhaltung der Baustein-Zuordnungsvorschrift folgendermaßen modelliert und anhand eines Beispiels verdeutlicht werden:

1. Zunächst wird während eines Chromosomendurchlaufs jeder Operation ein geeigneter Bausteintyp zugewiesen, indem eine Instanz k als Bausteintyp $m = type(k)$ interpretiert wird. Um die Operationen optimal konkreten Instanzen zuordnen zu können, ist es notwendig, daß die endgültige Zuweisung zu Instanzen erst dann vorgenommen wird, wenn die Anzahl der benötigten Instanzen jedes Bausteintyps feststeht. Das ist allerdings frühestens *nach* einem vollständigen Chromosomendurchlauf der Fall.

   Aus dem Startzeitpunkt $CS(j)$ und der Latenzzeit $\ell(j,k)$ ergeben sich die Zeitpunkte, zu denen eine Instanz für Operation j zur Verfügung gestellt werden muß. Zur Verwaltung der in einem Kontrollschritt benötigten Instanzen eines Bausteintyps wird die Tabelle tab_{alloc} benutzt. In dieser Tabelle wird bei Bedarf die Anzahl der im entsprechenden Kontrollschritt benötigten Instanzen des Bausteintyps erhöht. Nach der letzten Eintragung in diese Tabelle kann die Anzahl der benötigten Instanzen eines Bausteintyps berechnet werden, indem die maximale Anzahl von Operationen bestimmt wird, die im selben Kontrollschritt eine Instanz dieses Typs belegen.

2. Auf Basis der im vorherigen Schritt ermittelten Anzahl benötigter Instanzen eines Bausteintyps werden jetzt alle Operationen nach Startzeitpunkten sortiert und mittels Left-Edge-Algorithmus [KP87] den endgültigen Instanzen zugewiesen. Anhand der auf dem Chromosom kodierten Instanzen der Operationen werden dann die Eintragungen in die Tabelle tab_{bind} vorgenommen. Dabei können die Operationen zwar eventuell einer anderen Instanz, aber nicht einem anderen Bausteintyp zugewiesen werden.

Am folgenden Beispiel wird deutlich, daß die Zuweisung mittels Left-Edge-Algorithmus notwendig ist, um optimale Ergebnisse für einen gegebenen Schedule und eine gegebene Allokation zu

[3]Auf diese Tabelle wird später noch konkreter eingegangen (siehe Abbildung 3.8).

erreichen. In Abbildung 3.5 ist dazu ein Datenflußgraph und das entsprechende Chromosom mit den Operationen $j_1, \ldots, j_7$ gegeben. Das Chromosom enthält die Genbelegungen der Operationen nach einem Chromosomendurchlauf. Die Operationen $j_1, j_3, \ldots, j_6$ stellen in diesem Beispiel Additionen, die Operationen j_2 und j_7 Multiplikationen dar. Eine Zuweisung der Additionen soll nur zu den Instanzen $k_1, \ldots, k_4$ von Bausteintyp m_1 und der Multiplikationen nur zu den Instanzen k_5 und k_6 von Bausteintyp m_2 möglich sein. Die Latenzzeit und Ausführungszeit für eine Addition auf Bausteintyp m_1 beträgt in diesem Beispiel 2 Kontrollschritte, die einer Multiplikation auf Bausteintyp m_2 3 Kontrollschritte.

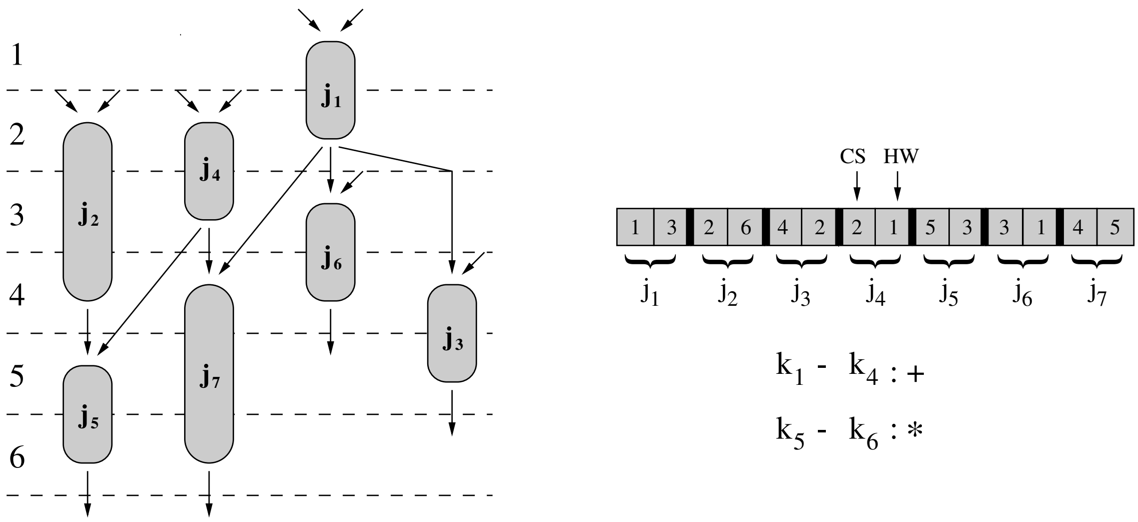


Abbildung 3.5: Datenflußgraph und Genbelegungen nach einem Chromosomendurchlauf

Wie in Abbildung 3.5 zu erkennen ist, wurde Operation j_1 während des Chromosomendurchlaufs der Kontrollschritt 1 und der Bausteintyp $m_1 = type(k_3)$ zugewiesen. Wie für die Operation j_1 werden ebenfalls die Instanzen der anderen Operationen vorerst nur als entsprechende Bausteintypen interpretiert. Aufgrund dieser Zuordnungen erfolgt nun die Berechnung der Anzahl benötigter Instanzen von jedem Bausteintyp, indem der maximale Eintrag einer Zeile in der tab_{alloc}-Tabelle bestimmt wird (siehe Abbildung 3.6).

Bausteintypen		1	2	3	4	5	6	7	# Instanzen
+ $k_1,\ldots,k_4$	m_1	1	2	2	2	2	1	0	**2**
* k_5, k_6	m_2	0	1	1	2	1	1	0	**2**

Kontrollschritte

Abbildung 3.6: Tabelle tab_{alloc} zur Bestimmung der benötigten Anzahl Instanzen für jeden Bausteintyp

In Abbildung 3.7 werden Eintragungen in Tabelle tab_{alloc} anhand der Operationen j_2 und j_7 betrachtet.

Da Operation j_2 in Kontrollschritt 2 gestartet wird und Bausteintyp $m_2 = type(k_6)$ aufgrund von $\ell(j_2, k_6) = 3$ erst nach 3 Kontrollschritten mit einer neuen Operation belegt werden darf, werden in der Tabelle tab_{alloc} die Einträge für die Kontrollschritte 2, 3 und 4 für m_2 erhöht (siehe Tabelle a). Mit den Einträgen für die Operation j_7 ergibt sich daraus für den Bausteintyp m_2 ein Gesamtbedarf von zwei Instanzen, da in Kontrollschritt 4 die Operationen j_2 und j_7 gleichzeitig eine Instanz dieses Bausteintyps benötigen (siehe Tabelle b).

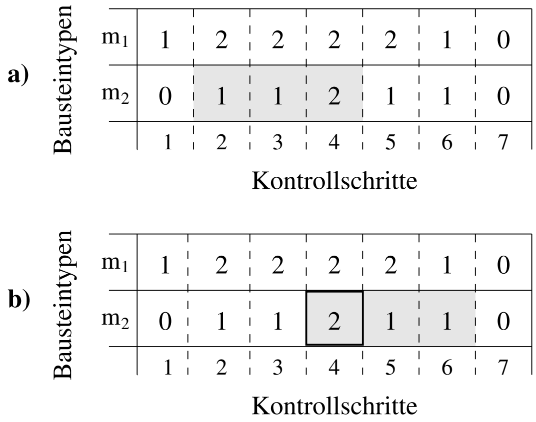


Abbildung 3.7: Einträge in Tabelle tab_{alloc}

O.B.d.A. werden immer die ersten Instanzen eines Bausteintyps belegt[4]. Die endgültige Zuordnung der Operationen zu Instanzen kann dabei auf unterschiedliche Arten erfolgen. In Abbildung 3.8 wurden die Operationen $j_1, \ldots, j_7$ in der Reihenfolge ihrer Ordnungsnummern in die Tabelle tab_{bind} eingetragen.

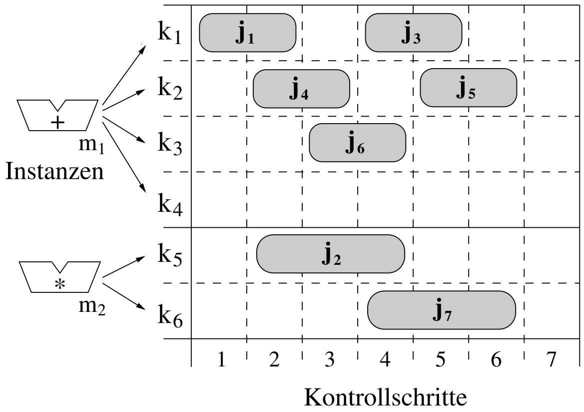


Abbildung 3.8: Tabelle tab_{bind} *ohne* Berücksichtigung der Ausführungsreihenfolge

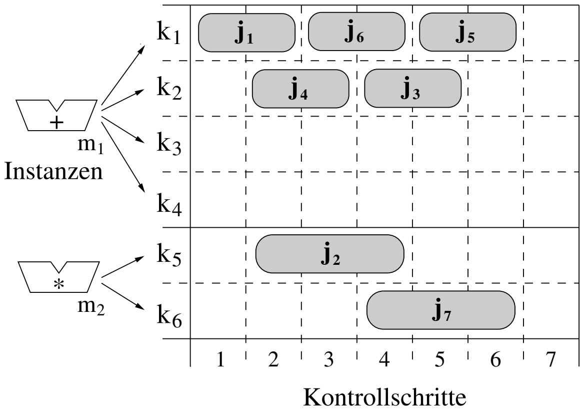


Abbildung 3.9: Tabelle tab_{bind} *mit* Berücksichtigung der Ausführungsreihenfolge

Es ist zu erkennen, daß bei dieser Vorgehensweise keine optimale Zuordnung zu Instanzen erreicht wird, da drei Instanzen von Bausteintyp m_1 benutzt werden, obwohl zwei Instanzen, wie in Abbildung 3.6 dargestellt, ausreichen. In Abbildung 3.9 ist das Zuweisungsergebnis unter der Verwendung des Left-Edge-Algorithmus zu sehen. Unter Berücksichtigung der Ausführungsreihenfolge der Operationen (j_1, j_4, j_2, j_6, j_3, j_7, j_5) zeigt sich, daß nun die Benutzung von zwei Instanzen für den Bausteintyp m_1 ausreicht. Die Zuweisungsreihenfolge der Operationen j_4 und j_2 sowie der Operationen j_3 und j_7 ist beliebig, da deren Ausführung jeweils im selben Kontrollschritt beginnt. Änderungen bei der Zuweisung von Instanzen werden für jede Operation als neues Allel des Hardware-Gens auf dem Chromosom gespeichert (siehe Abbildung 3.10). Im Vergleich zum Chromosom in Abbildung 3.5 sind lediglich Unterschiede bezüglich der Belegungen der Hardware-Gene vorhanden.

[4]Als zusätzliche Vorschrift zur Baustein-Zuordnungsvorschrift, wird durch die Instanz-Zuordnungsvorschrift [LMMD97] festgelegt, daß immer die ersten Instanzen eines Bausteintyps ausgewählt werden. Diese Vorschrift wird bei dieser Art der Zuweisung der Operationen zu Instanzen mitberücksichtigt. Die Instanz-Zuordnungsvorschrift führt bei der Lösung des linearen Gleichungssystems mit IP-Solvern zu einer schnelleren Berechnung der Lösung, hat aber keinen Einfluß auf die Qualität des Ergebnisses.

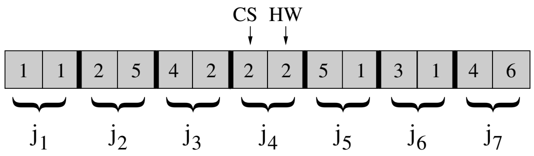


Abbildung 3.10: Endgültige Chromosomenbelegung

3.4.3 Vorrangsvorschrift

Durch die Vorrangsvorschrift werden die im Datenflußgraphen enthaltenen Datenabhängigkeiten zwischen den Operationen berücksichtigt. Zwischen zwei Operationen j und j' existiert eine Datenabhängigkeit $j \prec j'$, falls j' das Ergebnis von j als Eingabe benötigt. Mit der Ausführung von Operation j' darf in diesem Fall nicht vor Beendung von Operation j begonnen werden.

Die Berücksichtigung einer Vorrangsvorschrift $j \prec j'$ zwischen zwei beliebigen Operationen j und j' ist im genetischen Algorithmus zu dem Zeitpunkt möglich, zu dem der Startzeitpunkt $i = CS(j)$ und der Bausteintyp $m = type(k)$ $(k = HW(j))$ von Operation j bekannt sind. Um eine gültige Lösung zu erhalten, darf mit der Ausführung von Operation j' nicht vor Kontrollschritt $CS(j) + C(j,k)$ begonnen werden.
Aufgrund der bereits erwähnten topologischen Ordnung der Operationen ist eine Behandlung der Vorrangsvorschriften zwischen den Operationen während eines Chromosomendurchlaufs möglich. So gilt für alle Vorrangsvorschriften $j \prec j'$, daß die Operation j' im Chromosom hinter Operation j steht und deswegen auch später einem Kontrollschritt zugewiesen wird. Nach der Zuweisung von Operation j zu einem Kontrollschritt und einem Hardware-Baustein wird der frühestmögliche Ausführungszeitpunkt $\text{ASAP}_{dyn}(j')$ von Operation j' auf $CS(j) + C(j,k)$ hochgesetzt, falls $\text{ASAP}_{dyn}(j')$ zu diesem Zeitpunkt noch eine Zuweisung von Operation j' zu einem früheren Kontrollschritt erlaubt.
Da von mehreren Operationen aus Datenabhängigkeiten zu Operation j bestehen können, existiert für jede Operation j eine Liste $succ_{list}(j)$ mit allen direkten Nachfolgeoperationen, die jeweils vollständig abgearbeitet wird. Eine Änderung von $\text{ASAP}_{dyn}(j')$ der Nachfolgeoperation j' ergibt sich dann nach folgender Vorschrift:

$$\text{ASAP}_{dyn}(j') = \begin{cases} CS(j) + C(j,k) & \text{, falls} \quad CS(j) + C(j,k) \; > \; \text{ASAP}_{dyn}(j') \\ \text{ASAP}_{dyn}(j') & \text{, sonst} \end{cases}$$

Diese Vorschrift ist äquivalent mit folgendem Ausdruck, wobei max das Maximum zweier Werte bestimmt:

$$\text{ASAP}_{dyn}(j') = max\ (\text{ASAP}_{dyn}(j')\ ,\ CS(j) + C(j,k)) \tag{3.1}$$

Für den Fall, daß $C(j,k) = 0$ cs ist, kann Chaining durchgeführt werden, da eine Ausführung der Operationen j und j' im gleichen Kontrollschritt ermöglicht wird (siehe Abschnitt 3.4.4).

In Abbildung 3.11 wird diese Vorgehensweise an einem Beispiel verdeutlicht. Die Ausführung von Operation j beginnt in Kontrollschritt 4 auf Instanz k (siehe Abbildung 3.11 (1)). Da die Ausführungsdauer von Operation j auf Instanz k 3 Kontrollschritte beträgt, darf Operation j' frühestens in Kontrollschritt 7 ausgeführt werden. Eine Zuweisung zu Kontrollschritt 6 ist nach der Korrektur von $\text{ASAP}_{dyn}(j')$ auf Kontrollschritt 7 nun nicht mehr möglich (siehe Abbildung 3.11 (2)).

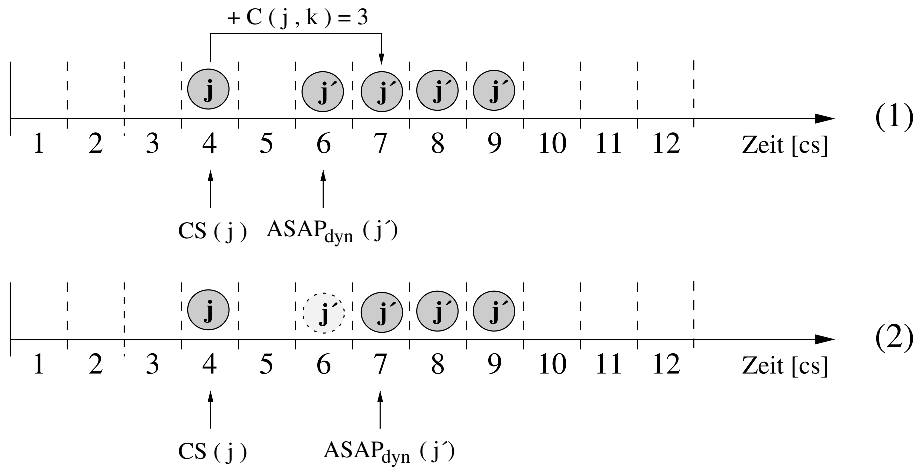


Abbildung 3.11: Einhaltung der Vorrangsvorschrift

Es zeigt sich an diesem Beispiel, daß durch die dargelegte Vorgehensweise die Verwendung mehrzyklischer Bausteintypen unter Einhaltung der Vorrangsvorschriften ermöglicht wird. Des weiteren können in Verbindung mit der Baustein-Zuordnungsvorschrift ebenfalls Pipeline-Bausteine benutzt werden. Wenn Operation j auf Instanz k ausgeführt wird, gilt in diesem Fall in der Regel: $\ell(j,k) < C(j,k)$.

Nachdem die Einhaltung der Basisvorschriften beschrieben worden ist, erfolgt in den nächsten Abschnitten die Beschreibung der optionalen Vorschriften.

3.4.4 Chaining

Die Durchführung von Chaining ermöglicht für zwei datenabhängige Operationen j und j' prinzipiell die Ausführung beider Operationen innerhalb eines Kontrollschrittes. Für die Operationen j und j' wird dies symbolisch durch $j \lll j'$ ausgedrückt. Durch die Verkettung von datenabhängigen Operationen wird in der Regel eine Reduzierung der Gesamtausführungszeit erreicht, da die Zykluszeit des Systemtaktes t_{cs} von den Operationen besser ausgenutzt werden kann[5]. Die Berücksichtung von Chaining kann allerdings auch zu höheren Kosten führen, da potentiell mehr Operationen in einem Kontrollschritt bearbeitet werden und eine Instanz in einem Kontrollschritt nicht mehrfach benutzt werden kann. Eine Voraussetzung zur Verkettung datenabhängiger Operationen ist, daß die Operationen auf Instanzen von Bausteintypen ausgeführt werden, die eine Ausführung innerhalb der Zykluszeit ermöglichen. Angaben zur physikalischen Ausführungszeit einer Operation j auf der Instanz k erfolgen mit $t_{chain}(j,k)$ in ns. Werden die Operationen j und j' zum Beispiel auf den Instanzen k und k' ausgeführt, so darf die Summe ihrer Ausführungszeiten die Zykluszeit nicht überschreiten. Es muß also $t_{chain}(j,k) + t_{chain}(j',k') \;\leq\; t_{cs}$ gelten.

In Abbildung 3.12 sind die drei Datenflußgraphen a, b und c dargestellt, an denen im folgenden das Grundprinzip von Chaining erklärt wird.

[5]Diese und folgende Angaben zur Zykluszeit des Systemtaktes t_{cs} stellen um Kontrollverzögerungen bereinigte Werte dar. Ebenso werden bei Angaben zur physikalischen Ausführungszeit $t_{chain}(j,k)$ für Instanz k bei Ausführung von Operation j bereits Verbindungsverzögerungen berücksichtigt.

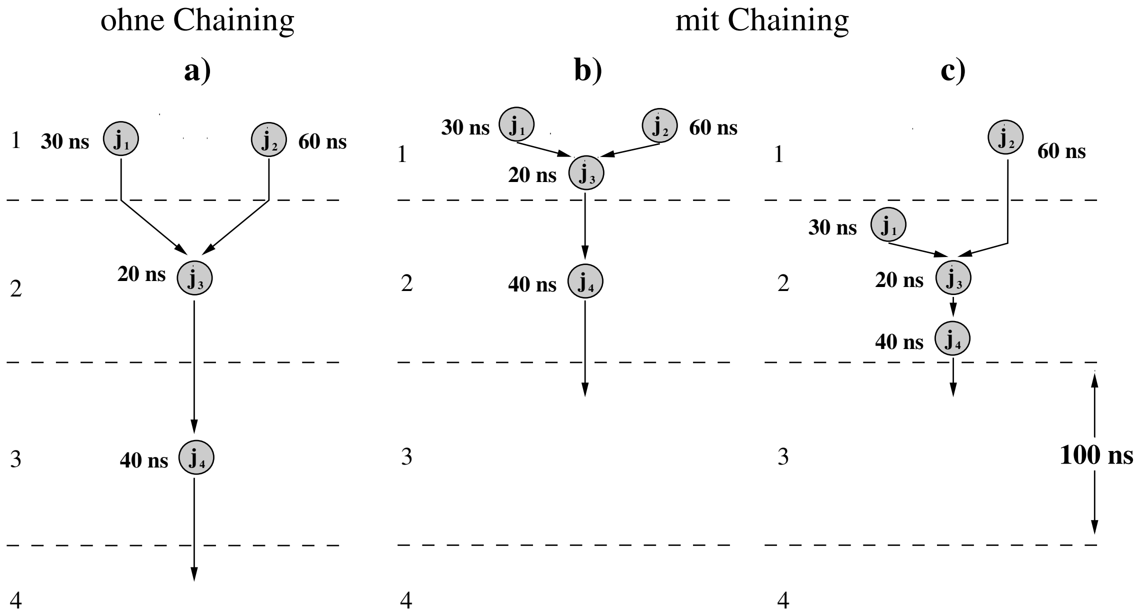


Abbildung 3.12: Datenflußgraphen ohne und mit Chaining

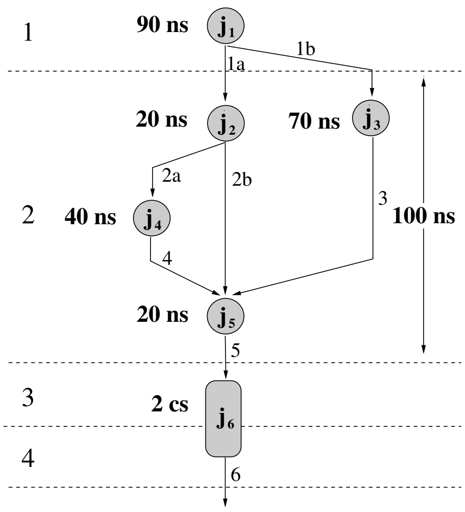
Es wird gezeigt, daß die Verkettung datenabhängiger Operationen in den Datenflußgraphen b und c zu einem Geschwindigkeitsvorteil gegenüber Datenflußgraph a führt, der kein Chaining von Operationen berücksichtigt. Zur Vereinfachung wird angenommen, daß den Operationen j_1 bis j_4 bereits Bausteintypen zugeordnet wurden. Die sich daraus ergebenden Ausführungszeiten der Operationen sind neben den jeweiligen Operationen angegeben. Für die Operationen bestehen folgende Chaining-Möglichkeiten: $j_1 \twoheadleftarrow j_3$, $j_2 \twoheadleftarrow j_3$ und $j_3 \twoheadleftarrow j_4$. Die in diesem Beispiel maximale realisierbare Operationen-Kette ergibt sich aus der Verkettung der Operationen j_1, j_3 und j_4 mit einer Gesamtausführungszeit von insgesamt $30\ ns + 20\ ns + 40\ ns = 90\ ns$ (siehe Datenflußgraph c). Die Realisierung einer Kette der Operationen j_2, j_3 und j_4 ist nicht möglich, da die Gesamtausführungszeit von 120 ns die Zykluszeit t_{cs} von 100 ns überschreiten würde. In dem Datenflußgraphen b kann aus diesem Grund die Operation j_4 nicht mehr in Kontrollschritt 1 ausgeführt werden und bildet so den Kopf einer neuen Kette in Kontrollschritt 2.
Für die Verkettung zweier Operationen j und j' ist es erforderlich, daß zum einen in der Vorrangsvorschrift für diese Operationen die Ausführung im selben Kontrollschritt erlaubt wird und zum anderen die Hintereinanderausführung der beiden Operationen innerhalb der Zykluszeit abgeschlossen ist. Falls die Operationen j und j' auf den Instanzen k und k' ausgeführt werden, gilt $C(j,k) = 0$ cs und $C(j',k') = 0$ cs. Die Latenzzeit der Instanzen k und k' beträgt jeweils einen Kontrollschritt, da die benutzten Instanzen erst nach Beendung des Zyklus mit neuen Operationen belegt werden dürfen.

Die Auswahl eines Kontrollschrittes für eine Operation wird immer anhand der *dynamischen* ASAP/ALAP-Bereiche der jeweiligen Operationen vorgenommen. Bei der Berücksichtigung von Chaining ist es nun allerdings möglich, daß die Zuweisung einer Operation j zum frühestmöglichen dynamischen Kontrollschritt $\mathrm{ASAP}_{dyn}(j)$ zu einer zwangsläufig ungültigen Lösung führt. Das kann der Fall sein, wenn bereits Operationen, zu denen eine Datenabhängigkeit existiert, in diesem Kontrollschritt ausgeführt werden. Dadurch besteht die Möglichkeit, daß kein Bausteintyp vorhanden ist, der eine ausreichend schnelle Ausführung der Operation innerhalb der verbliebenen Restausführungszeit in diesem Kontrollschritt erlaubt. Falls also die Ausführung von Operation j in Kontrollschritt $\mathrm{ASAP}_{dyn}(j)$ nicht innerhalb der Zykluszeit beendet werden kann, muß dieser Kontrollschritt als Ausführungszeitpunkt für diese Operation ausgeschlossen werden. Es ist also notwendig, daß für eine beliebige Operation j die Restausführungszeit in Kontrollschritt $\mathrm{ASAP}_{dyn}(j)$ effizient berechnet werden kann. Dazu wird für jede Operation j ein Akkumulator $ch_{accu}(j)$ verwaltet, der die kumulierte Ausführungszeit der längsten[6] Operationen-Kette in Kontrollschritt $\mathrm{ASAP}_{dyn}(j)$ für Operation j enthält. Die Berechnung der verbliebenen Restausführungszeit $t_{remain}(j, \mathrm{ASAP}_{dyn}(j))$ für Operation j im frühestmöglichen Ausführungszeitpunkt $\mathrm{ASAP}_{dyn}(j)$ ergibt sich dann aus der Differenz der Zykluszeit t_{cs} und dem Akkumulator $ch_{accu}(j)$ von Operation j. Es gilt:

$$t_{remain}(j, \mathrm{ASAP}_{dyn}(j)) = t_{cs} - ch_{accu}(j)$$

In allen späteren Ausführungszeitpunkten entspricht die verbliebene Restausführungszeit dem maximalen Wert, also der Zykluszeit t_{cs} des Systems.

Die Aktualisierungen des Akkumulators für jede einzelne Operation stellen die Grundlage zur Ausnutzung der Chaining-Möglichkeiten dar und werden bei der Behandlung der Vorrangsvorschriften während eines Chromosomendurchlaufs mitberücksichtigt. In Abbildung 3.13 ist ein Datenflußgraph, bestehend aus den sechs Operationen j_1 bis j_6, angegeben. Im folgenden wird die veränderte Abarbeitung der Operationen während eines Chromosomendurchlaufs anhand dieses Beispiels verdeutlicht.



Op	1		2		3		4		5		6	
	cs	accu	cs	accu	cs	accu	cs	accu	cs	accu	cs	accu
	1	0	1	0	1	0	1	0	1	0	1	0
1a				90								
1b						90						
2a			2	0			2	20				
2b									2	20		
3					2	0				70		
4												
5											2	90
6											3	0

Abbildung 3.13: Änderungen von ASAP_{dyn} und ch_{accu} bei der Berücksichtigung von Chaining

In der zugehörigen Tabelle werden für alle Operationen (Op) die Änderungen der ASAP_{dyn}-Werte (cs) und der Akkumulatoren (accu) wiedergegeben. Die Numerierungen an den Kanten des Datenflußgraphen geben die Abarbeitungsfolge der Vorrangsvorschriften an und bestimmen damit die Reihenfolge, in der die Chaining-Vorschriften berücksichtigt werden. Zu Beginn eines Chromosomendurchlaufs enthalten alle Akkumulatoren stets den Wert 0. Im Verlauf des Chromosomendurchlaufs werden die Operationen entsprechend ihrer Ordnung auf dem Chromosom besucht und in diesem Beispiel dem frühestmöglichen Kontrollschritt zugewiesen. Zur Verdeutlichung der Vorgehensweise wird angenommen, daß alle Operationen prinzipiell in Kontrollschritt 1 ausgeführt werden können. Im Datenflußgraphen ist neben jeder Operation die Ausführungszeit des *schnellsten* Bausteintyps angegeben. Anhand dieser Ausführungszeit kann für die entsprechende Operation überprüft werden, ob diese zum aktuell frühesten Ausführungszeitpunkt ausgeführt werden kann. Es wird vereinfachend angenommen, daß jede Operation

[6] „Längste Operationen-Kette“ bezieht sich auf die Ausführungszeit in ns und nicht auf die Anzahl der Operationen in der Kette.

auch auf einer Instanz dieses Bausteintyps ausgeführt wird. Eine Besonderheit stellt in diesem Beispiel die Operation j_6 dar, da mit dieser Operation aufgrund der Ausführungszeit von 2 Kontrollschritten kein Chaining mit anderen Operationen möglich ist.
Während eines Chromosomendurchlaufs werden die Operationen nun in der Reihenfolge ihrer Ordnungsnummern besucht. Dabei wird zur Einhaltung der Vorrangsvorschriften für jede aktuell zu behandelnde Operation j die Liste mit allen direkten Nachfolgeoperationen $succ_{list}(j)$ abgearbeitet. Die Bearbeitung erfolgt entsprechend der Numerierung an den Kanten. Dabei muß für jede in eine Operation j' eingehende Kante der Akkumulator $ch_{accu}(j')$ aktualisiert werden. In den unteren Zeilen der Tabelle werden zu jeder Vorrangsvorschrift entsprechend der Kantenbezeichnung die notwendigen Änderungen angegeben. Zum Beispiel werden für die Vorrangsvorschrift $j_3 \prec j_5$ in der Zeile 3 insgesamt drei Änderungen durchgeführt. Dies sind Änderungen, die sich auf die Operation j_3 selbst (siehe Spalte 3) und andererseits auf die Nachfolgeoperation j_5 auswirken (siehe Spalte 5).

Die Bearbeitung einer Vorrangsvorschrift $j \prec j'$ zwischen den Operationen j und j' wird folgendermaßen in den Arbeitsablauf integriert:

1. Überprüfung, ob genügend Restausführungszeit $t_{remain}(j, \text{ASAP}_{dyn}(j))$ vorhanden ist, um Operation j in Kontrollschritt $\text{ASAP}_{dyn}(j)$ auf einer Instanz eines geeigneten Bausteintyps auszuführen. Ist nicht genügend Restausführungszeit vorhanden, wird $\text{ASAP}_{dyn}(j)$ um 1 erhöht und $ch_{accu}(j)$ auf 0 zurückgesetzt. Dieser Arbeitsschritt wird für jede Operation genau einmal durchgeführt, da nach diesem Schritt der Ausführungszeitpunkt von Operation j festgelegt wird.

2. Änderung von $ch_{accu}(j')$ der direkten Nachfolgeoperation j'. Dabei muß zum einen der Fall betrachtet werden, daß das Ergebnis von Operation j innerhalb eines Kontrollschrittes vorliegt und zum anderen die Möglichkeit, daß die Ausführung der Operation j länger als einen Kontrollschritt dauert[7].

3. Änderung von $\text{ASAP}_{dyn}(j')$ der direkten Nachfolgeoperation j' (mit Formel (3.1)).

Zur Verdeutlichung werden im folgenden anhand des in Abbildung 3.13 dargestellten Beispiels diese drei Schritte näher betrachtet. Dazu werden für Schritt 1 die Kanten angegeben, *vor* deren Betrachtung eine Änderung von $\text{ASAP}_{dyn}(j)$ aufgrund zu geringer Restausführungszeit vorgenommen werden muß. In den Schritten 2 und 3 werden dann die Kanten angegeben, bei deren Abarbeitung entweder eine Änderung eines Akkumulators oder eines dynamischen ASAP-Wertes erforderlich ist. Diese beiden Schritte müssen für *jede* Vorrangsvorschrift $j \prec j'$ durchgeführt werden.

Zu Schritt 1: (Überprüfung der Restausführungszeit)
Vor der Abarbeitung von Kante 2a, d. h. der Vorrangsvorschrift $j_2 \prec j_4$, wird in diesem Schritt festgestellt, daß aufgrund der zu geringen Restausführungszeit von $100\ ns - 90\ ns = 10\ ns$ die Operation j_2 nicht mehr in Kontrollschritt 1 ausgeführt werden kann. Aus diesem Grund wird der frühestmögliche Ausführungszeitpunkt $\text{ASAP}_{dyn}(j_2)$ auf Kontrollschritt 2 hochgesetzt und der Akkumulator $ch_{accu}(j_2)$ von Operation j_2 auf $0\ ns$ heruntergesetzt (siehe Tabelleneintrag Zeile 2a, Spalte 2). Der Operation j_2 steht nun wieder genügend Restausführungszeit zur Verfügung.
Weitere Beispiele sind für die Operationen j_3 (Kante 3) und j_6 (Kante 6) gegeben und können ebenfalls der Tabelle entnommen werden.

[7] Eine Änderung von $ch_{accu}(j')$ wird im ersten Fall mit Formel (3.2) und im zweiten Fall mit Formel (3.3) auf den Seiten 28 f vorgenommen.

Zu Schritt 2: (Änderung von $ch_{accu}(j')$)
Die Bearbeitung von Kante 3 erfordert in diesem Schritt eine Korrektur des Akkumulators von Operation j_5. Nachdem die Operation j_3 zu Kontrollschritt 2 und einer Instanz eines Bausteintyps mit 70 ns Ausführungszeit zugewiesen worden ist, wird überprüft, ob der Akkumulator von Operation j_5 erhöht werden muß. Der aktuelle Akkumulatorstand von Operation j_5 beträgt zu diesem Zeitpunkt 20 ns (siehe Tabelleneintrag in Zeile 2b und Spalte 5), da die Vorrangsvorschrift $j_4 \prec j_5$ (siehe Kante 4) noch nicht betrachtet wurde und deswegen noch keinen Einfluß auf die Operation j_5 hat. Operation j_5 kann aufgrund von $\mathrm{ASAP}_{dyn}(j_5) = 2$ prinzipiell eine Verlängerung der beiden einelementigen Operationen-Ketten j_2 und j_3 darstellen. Aus diesem Grund muß $ch_{accu}(j_5)$ auf die maximale Ausführungsdauer der beiden Ketten gesetzt werden. Das entspricht genau dem Maximum des aktuellen Eintrages von $ch_{accu}(j_3)$ zuzüglich der Ausführungszeit von Operation j_3 und dem aktuellen Wert von $ch_{accu}(j_5)$. Durch die Anwendung der dritten Möglichkeit von Formel (3.2) wird also $ch_{accu}(j_5)$ auf $max(0\ ns + 70\ ns, 20\ ns) = 70\ ns$ gesetzt. Analog hierzu erfolgt die Abarbeitung der Kanten 1a und 1b.
Bei der Betrachtung der Kanten 2a, 2b und 5 werden die Akkumulatoren der abhängigen Operationen durch den zweiten Teil von Formel (3.2) geändert, da nach der Einhaltung der Vorrangsvorschrift im folgenden Schritt keine Chaining-Möglichkeit mit bereits betrachteten datenabhängigen Operationen mehr möglich sein wird.
Die Existenz einer weiteren Vorrangsvorschrift $j_6 \prec j_7$ würde in diesem Schritt durch die Anwendung von Formel (3.3) dazu führen, daß $ch_{accu}(j_7)$ auf 0 zurückgesetzt wird.

Zu Schritt 3: (Änderung von $\mathrm{ASAP}_{dyn}(j')$)
Die Berücksichtigung der Kanten 2a, 2b und 5 macht eine Erhöhung der ASAP_{dyn}-Werte um 1 für die entsprechenden Nachfolgeoperationen erforderlich.

Im folgenden werden die beiden Formeln angegeben, mit denen bei einer Vorrangsvorschrift $j \prec j'$ der Akkumulator $ch_{accu}(j')$ der Nachfolgeoperation j' geändert wird. Dazu wird neben der Möglichkeit, die beiden Operationen j und j' im selben Kontrollschritt auszuführen, auch berücksichtigt, daß aufgrund einer zu langen Ausführungszeit $C(j,k) \geq 1$ cs von Operation j auf Instanz k kein Chaining zwischen den Operationen j und j' realisiert werden kann. Es werden zwei Fälle unterschieden:

1. $C(j,k) = 0$: Das Ergebnis von Operation j liegt nach weniger als einem Kontrollschritt bei der Ausführung auf Instanz k vor. Es ist also prinzipiell ein Chaining zwischen den Operationen j und j' möglich. Weiterhin können noch Chaining-Möglichkeiten zwischen j' und anderen Vorgängeroperationen bestehen. Daraus ergeben sich folgende drei Unterfälle zur Änderung von $ch_{accu}(j')$:

$$ch_{accu}(j') =$$
$$\begin{cases} ch_{accu}(j') & \text{, falls } CS(j) < \mathrm{ASAP}_{dyn}(j') \\ ch_{accu}(j) + t_{chain}(j,k) & \text{, falls } CS(j) > \mathrm{ASAP}_{dyn}(j') \\ max\ (ch_{accu}(j) + t_{chain}(j,k)\ ,\ ch_{accu}(j')) & \text{, falls } CS(j) = \mathrm{ASAP}_{dyn}(j') \end{cases} \tag{3.2}$$

Der erste Unterfall gibt an, daß zwischen den Operationen j und j' kein Chaining durchgeführt werden kann, da eine Ausführung der beiden Operationen im selben Kontrollschritt nicht mehr möglich ist. Die Operation j hat deswegen keinen Einfluß auf den Akkumulator von Operation j'.
Im zweiten Unterfall kann für die Operation j' dagegen keine andere Chaining-Möglichkeit als mit Operation j bestehen, da $\mathrm{ASAP}_{dyn}(j')$ aufgrund der Vorrangsvorschrift $j \prec j'$

durch Anwendung von Formel (3.1) auf $CS(j)$ hochgesetzt wird.
Der letzte Unterfall stellt eine Kombination der beiden vorherigen Möglichkeiten dar und berücksichtigt Chaining-Möglichkeiten sowohl mit Operation j als auch mit anderen Vorgängeroperationen. Es wird das Maximum der kumulierten Ausführungszeit der Operationen-Kette mit Operation j und, falls vorhanden, mit einer weiteren direkten Vorgängeroperation bestimmt.

2. $\boldsymbol{C(j,k) \geq 1}$ **:** Zwischen den Operationen j und j' ist in diesem Fall kein Chaining möglich, da die Ausführung von Operation j auf Instanz k mindestens einen Kontrollschritt benötigt. Um bisherige Chaining-Möglichkeiten von Operation j' korrekt zu berücksichtigen, darf $ch_{accu}(j')$ nur dann auf 0 zurückgesetzt werden, wenn $\text{ASAP}_{dyn}(j')$ aufgrund der Vorrangsvorschrift $j \prec j'$ hochgesetzt wird. Es ergibt sich:

$$ch_{accu}(j') = \begin{cases} 0 & \text{, falls} \quad CS(j) + C(j,k) \; > \; \text{ASAP}_{dyn}(j') \\ ch_{accu}(j') & \text{, sonst} \end{cases} \tag{3.3}$$

Durch beide Fälle wird für jede Operation j prinzipiell neben der Ausführung auf einer Instanz mit $t_{chain}(j,k) \leq t_{cs}$ ebenfalls die Ausführung auf Instanzen von ein- oder mehrzyklischen Bausteintypen ermöglicht.

3.4.5 Makrooperationen

Es gibt erweiterte Bausteinbibliotheken, in denen neben den einfachen Bausteintypen wie Addierer und Multiplizierer, komplexere Bausteintypen zur Verfügung gestellt werden. Ein Beispiel für Komplexbausteine stellt der Multiplier-Adder-Accumulator (MAC) dar. Dieser ist in der Lage, eine Multiplikation gefolgt von einer Addition als sogenannte Makrooperation auszuführen. Allgemein besteht eine Makrooperation aus der Zusammenfassung von mindestens zwei Operationen, die gemeinsam auf einem entsprechenden Komplexbaustein ausgeführt werden können.

In Abbildung 3.14 ist ein Beispiel für einen MAC-Baustein angegeben. Die Teilung des Datenflußgraphen verdeutlicht, daß nur *ein* Datenpfad a) oder b) in Hardware realisiert werden darf. Die Benutzung von Datenpfad a) bedeutet, daß die Makrooperation j_1 auf dem Bausteintyp m_1 (MAC) realisiert wird. Wird der Datenpfad b) ausgewählt, so werden die Einzeloperationen (Multiplikation und Addition) realisiert.

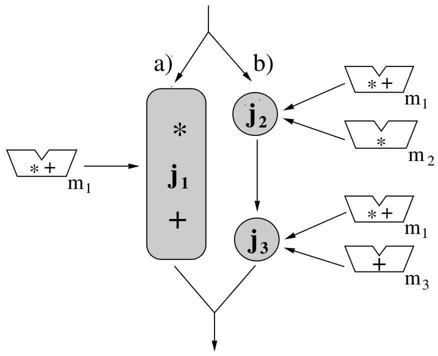


Abbildung 3.14: Beispiel für einen MAC

Komplexbausteine verursachen zwar in der Regel höhere Einzelkosten als einfache Bausteine, können allerdings durch die variable Einsatzweise die Gesamtkosten eines Entwurfs senken, indem nicht nur Makrooperationen, sondern ebenfalls einfache Operationen auf ihnen ausgeführt werden. Das ist jedoch nur bei solchen Bausteintypen möglich, bei denen die nicht benötigten Funktionseinheiten durch neutrale Elemente „mathematisch" ausgeschaltet werden können.

Zur Berücksichtigung von Makrooperationen in einem Entwurf ist eine Erweiterung der Operations-Zuordnungsvorschrift (siehe Abschnitt 3.4.1) erforderlich. Gegenüber einem Entwurf ohne Makrooperationen muß ein gegenseitiger Ausschluß einer Makrooperation und den dazugehörigen Einzeloperationen realisiert werden, damit nicht alle kodierten Operationen in Hardware umgesetzt werden. Zur Unterscheidung der Operationen wird mit jeder Operation $j \in J$ eine zusätzliche Markierung assoziiert und nach folgender Vorschrift belegt:

$$macro_{flag}(j) = \begin{cases} 1 & \text{, falls } j \text{ Makrooperation ist} \\ 2 & \text{, falls } j \text{ erste Einzeloperation ist} \\ n+1 & \text{, falls } j \text{ n-te Einzeloperation ist} \\ 0 & \text{, sonst} \end{cases} \tag{3.4}$$

Die Abbildung 3.15 verdeutlicht den gegenseitigen Ausschluß von Makrooperationen und dazugehörigen Einzeloperationen am Beispiel eines MAC-Bausteins.

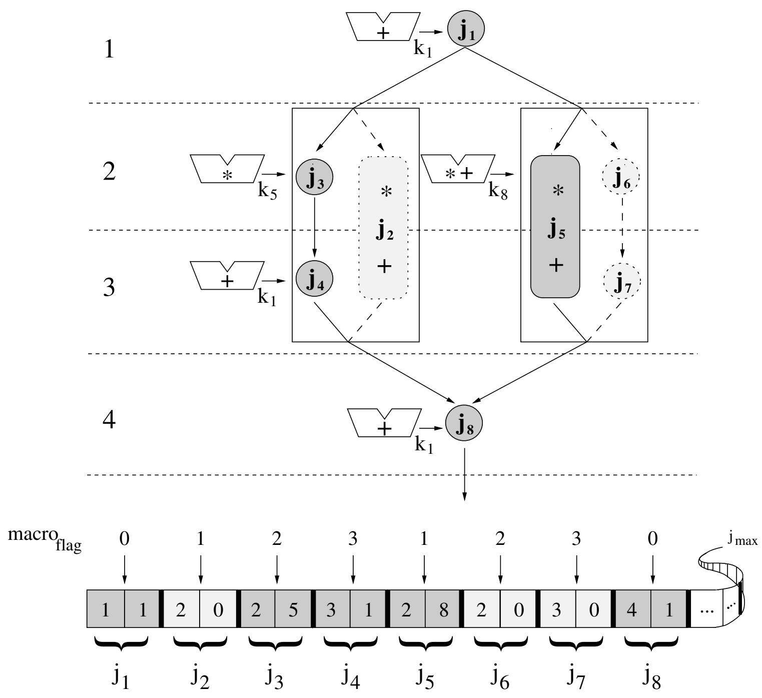


Abbildung 3.15: Berücksichtigung von Makrooperationen

In Abbildung 3.15 ist über dem Chromosom für jede Operation die entsprechende Markierung angegeben. In diesem Beispiel wird angenommen, daß die Operationen j_2, j_6 und j_7 nicht in Hardware realisiert werden. Sie werden auf dem Chromosom gekennzeichnet, indem das entsprechende Hardware-Gen mit dem Allel 0 belegt wird, das keine Interpretation als Hardware-Baustein zuläßt. Der gegenseitige Ausschluß ist einfach durchzuführen, da für jede Makrooperation j gilt, daß die dazugehörigen n Einzeloperationen $\in \{j+1, \ldots, j+n\}$ aufeinanderfolgende Ordnungsnummern besitzen und deswegen im Chromosom hintereinander stehen. Hierbei wird bei einem Chromosomendurchlauf bei Erreichen einer Operation mit $macro_{flag} = 1$ (hier: Operationen j_2 und j_5) entschieden, welche der beiden Alternativen realisiert wird.

3.4.6 Zeitvorgabevorschriften

Bei dem Entwurf eines Systems ist es wichtig, daß der Benutzer zur Realisierung von Timing-Protokollen Zeitvorgabevorschriften angeben kann, die für je zwei Operationen die zeitlichen

Ausführungsabstände festlegen. Das betrifft insbesondere Registeroperationen die zur Zwischenspeicherung von Werten benutzt werden. Es erfolgt hier eine Beschränkung auf Registeroperationen (beziehungsweise einzyklische Operationen), die innerhalb eines Kontrollschrittes ausgeführt werden können. Eine Erweiterung auf mehrzyklische Operationen ist jedoch möglich.
O.B.d.A. sei eine Zeitvorgabevorschrift zwischen zwei beliebigen Operationen j und j' durch den minimalen $min_{timing}(j, j')$ und den maximalen $max_{timing}(j, j')$ Zeitabstand genau festgelegt. Für den Fall, daß zwischen zwei Operationen nur einer der möglichen Zeitabstände vom Benutzer angegeben wird, kann der nicht spezifizierte in der Art ergänzt werden, daß keine weiteren Einschränkungen für die beiden Operationen j und j' erfolgen. Sollen nur minimale Zeitabstände zwischen den Operationen j und j' betrachtet werden, wird als maximaler Zeitabstand $max_{timing}(j, j')$ der maximal erlaubte Kontrollschritt i_{max} als Wert gesetzt. Offensichtlich stellt diese Belegung keine weitere Einschränkung für die Operationen j und j' dar. Analog dazu wird für maximale Zeitabstände entsprechend $min_{timing}(j, j')$ auf 0 gesetzt, so daß eine Ausführung von Operation j und j' prinzipiell im selben Kontrollschritt möglich ist.

Eine Zeitvorgabevorschrift zwischen den Operationen j und j' wird bei einem Chromosomendurchlauf bei Erreichen von Operation j berücksichtigt. Dazu wird für eine Operation j die Liste $timing_{list}(j)$ verwaltet, die alle Zeitvorgabevorschriften zu Operationen mit einer größeren Ordnungsnummer enthält. Die eingetragenen Operationen können aufgrund der größeren Ordnungsnummern zu diesem Zeitpunkt noch keinem Kontrollschritt zugewiesen worden sein.

Die Berücksichtigung einer Zeitvorgabevorschrift zwischen j und j' wird in zwei Phasen unterteilt, die zum einen Auswirkungen auf die Kontrollschritt-Grenzen von Operation j und zum anderen auf die von Operation j' haben:

1.) Zur Vermeidung ungültiger Lösungen ist eine Vorausschau auf mögliche Ausführungszeitpunkte von Operation j' sinnvoll. Dazu werden nachfolgend vier Fälle vorgestellt, mit denen einige Kontrollschritte als Belegungsmöglichkeit für Operation j ausgeschlossen werden, die zwangsläufig zu ungültigen Kontrollschritt-Belegungen für Operation j' führen würden. Diese Einschränkungen müssen dementsprechend *vor* der Zuordnung von Operation j zu einem Kontrollschritt vorgenommen werden. Diese Phase endet mit der Zuweisung von Operation j zu einem Kontrollschritt und stellt damit die Voraussetzung für die zweite Phase dar.

2.) *Nachdem* die Operation j einem Kontrollschritt zugewiesen wurde, müssen alle zu ungültigen Lösungen führenden Kontrollschritte von Operation j' als Belegungsmöglichkeiten ausgeschlossen werden.

Zu 1) Kontrollschritt-Einschränkungen für Operation j
Durch die Einschränkung der Kontrollschritt-Grenzen $\mathrm{ASAP}_{dyn}(j)$ und $\mathrm{ALAP}_{dyn}(j)$ lassen sich ungültige Belegungen des Kontrollschritt-Gens von Operation j vermeiden, die bei einer Auswahl zwangsläufig zu einer ungültigen Lösung führen würden, da kein gültiger Kontrollschritt für Operation j' mehr ausgewählt werden kann.

Zur Durchführung der Bereichseinschränkungen werden die folgenden Unterfälle unterschieden: In den Unterfällen (a) und (b) werden die dynamischen Grenzen der Operation j unabhängig von der Ausführungsreihenfolge der Operationen j und j' eingeschränkt und betreffen deswegen alle Zeitvorgabevorschriften. Die Unterfälle (c) und (d) werden bei einer feststehenden Ausführungsreihenfolge der Operationen angewendet. Falls Operation j *vor* Operation j' ausgeführt wird, ist dabei im Unterfall (c) eine Korrektur des spätestmöglichen Ausführungszeitpunktes von Operation j möglich. Analog dazu kann in Unterfall (d) der frühestmögliche Ausführungszeitpunkt von Operation j heraufgesetzt werden, falls j *nach* j' ausgeführt wird.

(a) (keine eindeutige Ausführungsreihenfolge)
Unabhängig von der Ausführungsreihenfolge der Operationen j und j' werden durch Hochsetzen von $\mathrm{ASAP}_{dyn}(j)$ diejenigen Kontrollschritte als Belegungen für Operation j ausgeschlossen, von denen trotz Ausnutzen des maximal erlaubten Zeitabstandes $\mathrm{ASAP}_{dyn}(j')$ nicht erreicht werden kann[8]. Es ergibt sich folgende Vorschrift:

$$\mathrm{ASAP}_{dyn}(j) = max\ (\mathrm{ASAP}_{dyn}(j)\ ,\ \mathrm{ASAP}_{dyn}(j') - max_{timing}(j, j'))$$

In Abbildung 3.16 ist ein Beispiel für die Korrektur von $\mathrm{ASAP}_{dyn}(j)$ dargestellt.

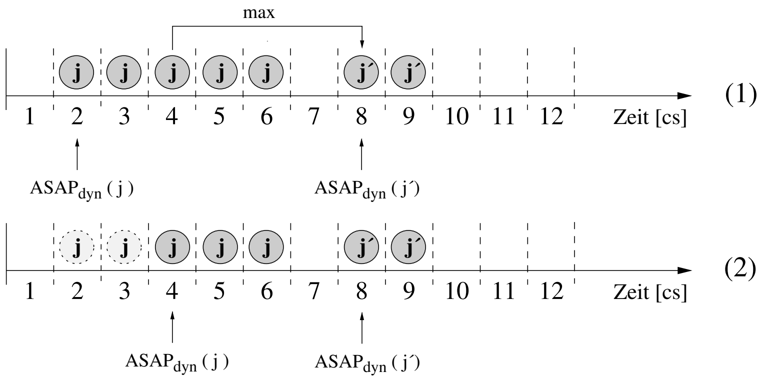


Abbildung 3.16: Korrektur von $\mathrm{ASAP}_{dyn}(j)$ bei beliebiger Ausführungsreihenfolge der Operationen j und j'

Der maximale Zeitabstand $max_{timing}(j, j')$ zwischen den Operationen j und j' soll in diesem Beispiel 4 Kontrollschritte betragen. Teil (1) der Abbildung verdeutlicht, daß bei der Auswahl von Kontrollschritt 2 oder 3 als Ausführungszeitpunkt für die Operation j keine gültige Belegung für die Operation j' mehr möglich ist. Die Korrektur von $\mathrm{ASAP}_{dyn}(j)$ auf Kontrollschritt 4 wird in Teil (2) der Abbildung dargestellt.

(b) (keine eindeutige Ausführungsreihenfolge)
In diesem Fall kann ein Herabsetzen von $\mathrm{ALAP}_{dyn}(j)$ erforderlich sein. In Abbildung 3.17 muß $\mathrm{ALAP}_{dyn}(j)$ von 9 auf 7 heruntergesetzt werden, da für Operation j' der spätestmögliche Ausführungzeitpunkt $\mathrm{ALAP}_{dyn}(j') = 3$ ist und mit dem maximalen Zeitabstand von $max_{timing}(j, j') = 4$ maximal der Kontrollschritt 7 erreicht werden kann.

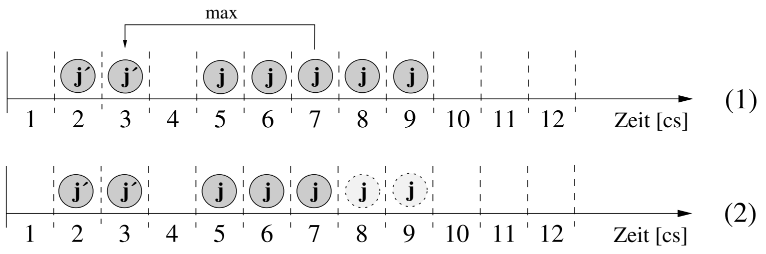


Abbildung 3.17: Korrektur von $\mathrm{ALAP}_{dyn}(j)$ bei beliebiger Ausführungsreihenfolge der Operationen j und j'

[8]In den Abbildungen werden entsprechende Zeitvorgaben verkürzt mit min und max bezeichnet, die nicht mit den Funktionen *min* und *max* zu verwechseln sind.

Es ergibt sich als allgemeine Vorschrift:

$$\mathrm{ALAP}_{dyn}(j) = min\ (\mathrm{ALAP}_{dyn}(j)\ ,\ \mathrm{ALAP}_{dyn}(j') + max_{timing}(j, j'))$$

(c) (j wird *vor* j' ausgeführt)
Die in diesem Unterfall betrachtete Einschränkung darf nur dann durchgeführt werden, wenn feststeht, daß die Operation j vor der Operation j' ausgeführt wird.

In Abbildung 3.18 werden die Operationen j und j' mit einem minimalen Zeitabstand von 4 Kontrollschritten betrachtet.

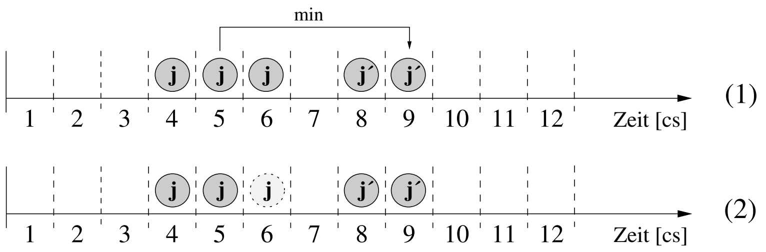


Abbildung 3.18: Korrektur von $\mathrm{ALAP}_{dyn}(j)$, bei feststehender Ausführungsreihenfolge der Operationen j und j'

Der Kontrollschritt $\mathrm{ALAP}_{dyn}(j) = 6$ wird als Belegung von Operation j ausgeschlossen, da dieser unter der Einhaltung des minimalen Zeitabstandes von 4 Kontrollschritten keine gültige Zuweisung von Operation j' zu einem Kontrollschritt erlauben würde.

Die Operation j wird in genau einem der beiden folgenden Fälle vor Operation j' ausgeführt:

1. $j \prec j'$ oder
2. $\mathrm{ALAP}_{dyn}(j) - min_{timing}(j, j') < \mathrm{ASAP}_{dyn}(j')$

In diesem Fall kann $\mathrm{ALAP}_{dyn}(j)$ heruntergesetzt werden, falls bei Berücksichtigung des minimalem Zeitabstand $min_{timing}(j, j')$, ausgehend von $\mathrm{ALAP}_{dyn}(j)$, der $\mathrm{ALAP}_{dyn}(j')$-Wert überschritten wird. Es ergibt sich die folgende Vorschrift:

$$\mathrm{ALAP}_{dyn}(j) = min\ (\mathrm{ALAP}_{dyn}(j)\ ,\ \mathrm{ALAP}_{dyn}(j') - min_{timing}(j, j'))$$

(d) (j wird *nach* j' ausgeführt)
Falls aufgrund der dynamischen ASAP- und ALAP-Bereiche der Operationen j und j' feststeht, daß Operation j nach Operation j' ausgeführt wird, ist eine Korrektur des frühestmöglichen Ausführungszeitpunktes $\mathrm{ALAP}_{dyn}(j)$ von Operation j möglich.

In Abbildung 3.19 ist ein Beispiel für eine solche Korrektur unter Berücksichtigung eines minimalen Zeitabstandes von 4 Kontrollschritten angegeben.

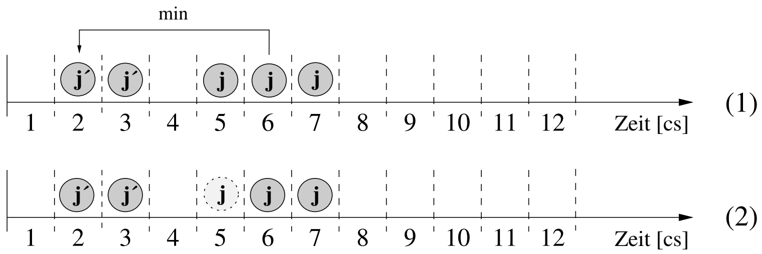


Abbildung 3.19: Korrektur von $\mathrm{ASAP}_{dyn}(j)$, bei feststehender Ausführungsreihenfolge der Operationen j und j'

Da ausgehend von $\mathrm{ASAP}_{dyn}(j') = 2$ mit dem minimalem Zeitabstand $\mathrm{ASAP}_{dyn}(j)$ von 4 Kontrollschritten bereits der Kontrollschritt $\mathrm{ASAP}_{dyn}(j) = 5$ überschritten wird, darf Operation j nicht Kontrollschritt 5 zugewiesen werden. Die Zuweisung von Operation j zu Kontrollschritt 5 wird durch Hochsetzen von $\mathrm{ASAP}_{dyn}(j)$ auf 6 vermieden.

Allgemein wird $\mathrm{ASAP}_{dyn}(j)$ in diesem Fall durch die folgende Vorschrift neu bestimmt:

$$\mathrm{ASAP}_{dyn}(j) = max\ (\mathrm{ASAP}_{dyn}(j)\ ,\ \mathrm{ASAP}_{dyn}(j') + min_{timing}(j, j'))$$

Aufgrund der topologischen Ordnung der Operationen kann keine Vorrangsvorschrift $j' \prec j$ existieren, da Operation j' eine höhere Ordnungsnummer als Operation j hat. In diesem Fall wird die Operation j nach Operation j' ausgeführt, falls der spätestmögliche Ausführungszeitpunkt von Operation j' abzüglich des minimal einzuhaltenden Zeitabstandes kleiner als der frühestmögliche Ausführungszeitpunkt von Operation j ist.

Formal ausgedrückt gilt:

$$\mathrm{ALAP}_{dyn}(j') - min_{timing}(j, j')\ <\ \mathrm{ASAP}_{dyn}(j)$$

Dies bedeutet, daß $\mathrm{ALAP}_{dyn}(j')$ von Operation j' zwar größer als $\mathrm{ASAP}_{dyn}(j)$ von Operation j sein kann, aber aufgrund des minimalen Zeitabstandes diese beiden Extremwerte den Operationen nicht als Kontrollschritte zugewiesen werden können.

Wenn die Ausführungsreihenfolge der am Timing beteiligten Operationen nicht eindeutig feststeht, kann es vorkommen, daß eine Einschränkung der unteren und oberen Kontrollschritt-Grenze von Operation j zur Vermeidung ungültiger Lösungen nicht ausreicht. Da einzelne Kontrollschritte innerhalb der Grenzen zu ungültigen Lösungen führen können, ist es möglich, daß Lücken entstehen, die nicht durch einfache Grenzeinschränkungen zu beseitigen sind. Dieser Spezialfall kann nur mit großem zusätzlichen Rechenaufwand berücksichtigt werden, indem jeder einzelne Kontrollschritt innerhalb der dynamischen Grenzen auf Gültigkeit getestet wird und zu ungültigen Lösungen führende Kontrollschritte gekennzeichnet werden. Es würde dadurch ein nicht vertretbarer Overhead entstehen. Da die in den Unterfällen (a) bis (d) betrachteten Bereichseinschränkungen von Operation j lediglich Vorsorgemaßnahmen zur Verhinderung ungültiger Lösungen darstellen, ist zudem eine erschöpfende Betrachtung an dieser Stelle nicht erforderlich.

Durch die im folgenden beschriebenen Bereichseinschränkungen von Operation j' müssen allerdings *alle* zu ungültigen Lösungen führenden Kontrollschritte als ungültig markiert werden. Ungültige Lösungen sind dann daran zu erkennen, daß für mindestens eine Operation keine gültige Kontrollschritt-Belegung möglich war. Die Durchführung der Bereichseinschränkungen kann unter Umständen dazu führen, daß für eine Operation j ASAP$_{dyn}(j)$ größer als ALAP$_{dyn}(j)$ gesetzt werden muß. In diesem Fall kann keine gültige Lösung mehr für dieses Individuum erreicht werden. Die Differenz der beiden Grenzen stellt allerdings ein gutes Fehlermaß dar und wird bei der Bewertung des entsprechenden Individuums berücksichtigt.

Zu 2) Kontrollschritt-Einschränkungen für Operation j'

Die Voraussetzung zur Korrektur der Kontrollschritt-Grenzen ASAP$_{dyn}(j')$ und ALAP$_{dyn}(j')$ ist, daß der Ausführungszeitpunkt $i = CS(j)$ von Operation j bereits feststeht. Auf der Basis des für die Operation j festgelegten Ausführungszeitpunktes können nun Kontrollschritt-Einschränkungen für die Operation j' durchgeführt werden.
Die Berücksichtigung von Zeitabständen zwischen zwei *datenabhängigen* Operationen j und j' kann durch Einschränkungen der ASAP$_{dyn}$- und ALAP$_{dyn}$-Grenzen geschehen, da Operation j' aufgrund der Vorrangsvorschrift $j \prec j'$ in jedem Fall nach der Operation j ausgeführt wird. Zur Einhaltung von minimalen Zeitabständen reicht es aus, bei Bedarf den ASAP$_{dyn}(j')$-Wert zu erhöhen, um ähnlich wie bei der Behandlung der Vorrangsvorschriften einen Mindestabstand zwischen zwei Operationen zu erzwingen. Entsprechend kann zur Einhaltung eines maximalen Zeitabstandes der ALAP$_{dyn}(j')$-Wert heruntergesetzt werden. Bei der Betrachtung von Zeitvorgabevorschriften zwischen zwei datenabhängigen Operationen können keine Lücken im Kontrollschritt-Bereich einer Operation auftreten, da zu ungültigen Lösungen führende Kontrollschritte immer nur an den jeweiligen oberen und unteren Grenzen auftreten können. Die Behandlung von datenabhängigen Operationen stellt also prinzipiell kein Problem dar.
Schwieriger ist dagegen die Berücksichtigung von Zeitvorgabevorschriften zwischen zwei Operationen j und j', falls keine Datenabhängigkeit vorhanden ist. In diesem Fall reicht es nicht immer aus, eine Einschränkung der dynamischen Grenzen von Operation j' vorzunehmen, da unter bestimmten Voraussetzungen Gültigkeitslücken im Kontrollschritt-Bereich zwischen ASAP$_{dyn}(j')$ und ALAP$_{dyn}(j')$ auftreten können. In Abbildung 3.20 ist dazu ein Beispiel für das Auftreten von Lücken, also ungültigen Kontrollschritten innerhalb der beiden dynamischen Grenzen von Operation j', angegeben.

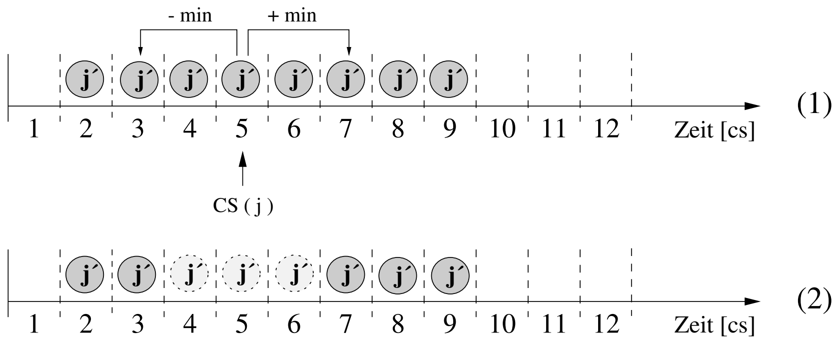


Abbildung 3.20: Auftreten von Lücken im Kontrollschritt-Array

Der Operation j wurde hier Kontrollschritt 5 zugewiesen. Für einen minimal einzuhaltenden Zeitabstand von 2 Kontrollschritten müssen die Kontrollschritte 4, 5 und 6 als Belegungen für Operation j' ausgeschlossen und als ungültig markiert werden. Im Gegensatz zu datenabhängigen Operationen müssen nun die Zeitvorgabevorschriften in zwei Richtungen betrachtet werden. Als Spezialfall kann es also vorkommen, daß bei bestimmten Zeitvorgaben Kontrollschritte innerhalb der beiden dynamischen Grenzen von Operation j' ausgeschlossen werden müssen. Für diese

Fälle wird für diese Operationen ein Array eingeführt, mit dem gezielt Kontrollschritte innerhalb der dynamischen Grenzen ungültig gesetzt werden können.
Bei einer Zeitvorgabe zwischen den Operation j und j' wird für Operation j' ein Kontrollschritt-Array $cs_{array}(j')$ unter den beiden folgenden Bedingungen benötigt:

a) Die Ausführungsreihenfolge der Operationen steht nicht eindeutig fest, d. h. es gilt:

   1) zwischen j und j' existiert keine Vorrangsvorschrift und

   2) (a) $\mathrm{ALAP}(j) - min_{timing}(j, j') \geq \mathrm{ASAP}(j')$ oder

      (b) $\mathrm{ASAP}(j) + min_{timing}(j, j') \leq \mathrm{ALAP}(j')$

b) Der minimal einzuhaltende Zeitabstand $min_{timing}(j, j')$ zwischen den Operationen j und j' ist größer als 0.

Bei der Behandlung einer Zeitvorgabe zwischen Operation j und j' ergeben sich für j' die folgenden Einschränkungen für $\mathrm{ASAP}_{dyn}(j')$ und $\mathrm{ALAP}_{dyn}(j')$:

1) Es erfolgt eine Korrektur der oberen dynamischen Grenze von Operation j', unabhängig von der Ausführungsreihenfolge, falls ausgehend vom gesetzten Kontrollschritt $i = CS(j)$ mit maximalem Zeitabstand $max_{timing}(j, j')$ die obere dynamische Grenze von Operation j' nicht erreichbar ist. Die neue obere dynamische Grenze von Operation j' ergibt sich aus:

$$\mathrm{ALAP}_{dyn}(j') = min\ (\mathrm{ALAP}_{dyn}(j')\ ,\ CS(j) + max_{timing}(j, j'))$$

2) Die Korrektur der unteren dynamischen Grenze von Operation j' bedarf einer Fallunterscheidung:

   a) Es existiert eine Datenabhängigkeit $j \prec j'$:

$$\mathrm{ASAP}_{dyn}(j') = max\ (\mathrm{ASAP}_{dyn}(j')\ ,\ CS(j) + min_{timing}(j, j'))$$

   b) Es existiert keine Datenabhängigkeit zwischen den Operationen j und j':

$$\mathrm{ASAP}_{dyn}(j') = max\ (\mathrm{ASAP}_{dyn}(j')\ ,\ CS(j) - max_{timing}(j, j'))$$

   Für diesen Fall muß die Möglichkeit betrachtet werden, daß Gültigkeitslücken im Kontrollschritt-Bereich auftreten können. Das kann nur der Fall sein, wenn $min_{timing}(j, j')$ größer als 0 ist (siehe Abbildung 3.20).

In diesem Abschnitt wurde gezeigt, in welcher Weise Zeitvorgabevorschriften zwischen den Operationen berücksichtigt werden können. Insbesondere besteht eine effiziente Möglichkeit zur Behandlung von Zeitvorgabevorschriften zwischen Operationen, für die untereinander keine Vorrangsvorschrift besteht.

3.4.7 Verbindungsminimierung

Die Durchführung der Verbindungsminimierung führt zu einer Erweiterung der Kostenfunktion (siehe Abschnitt 3.3) und kann vom Benutzer optional angegeben werden. Das Ziel der Verbindungsminimierung besteht darin, die Kosten der benötigten Verbindungen zwischen den Hardware-Bausteinen so weit wie möglich zu reduzieren. Bei einer Datenabhängigkeit von Operation j' gegenüber der Operation j wird eine Verbindung zwischen den Instanzen k und k' benötigt, falls die Operation j auf der Instanz k und die Operation j' auf der Instanz k' ausgeführt wird. In diesem Fall liegt also ein Datenfluß von Instanz k zu k' vor. Falls die Operationen j und j' auf derselben Instanz k ausgeführt werden, ist eine Verbindung des Ausgangsports mit einem Eingangsport der Instanz k erforderlich. Da die Bestimmung der Verbindungskosten für einen Entwurf erst nach der Zuordnung der Operationen zu konkreten Instanzen möglich ist und gegenseitige Abhängigkeiten zwischen Scheduling, Allokation und Ressourcen-Bindung bestehen, ist eine gleichzeitige Betrachtung der Teilaufgaben notwendig. Mit jeder benötigten Verbindung $w_{k,k'}$ zwischen einer Instanz k und k' werden die durch diese Verbindung entstandenen Kosten von $c_{k,k'}$ zu den Gesamtkosten des Entwurfs addiert.
Eine Integration der Verbindungsminimierung in den genetischen Algorithmus stellt kein Problem dar, da sie keinen Einfluß auf die Gültigkeit von Lösungen hat. Durch eine ungünstige Bindung der Operationen an die Instanzen entsteht im schlimmsten Fall ein weniger kostengünstiges Individuum, aber in keinem Fall eines, das nicht den spezifizierten Nebenbedingungen genügt.
Zur Reduzierung der Verbindungskosten ist es erforderlich, die Vorgehensweise bei der Ressourcen-Bindung so zu verändern, daß vorhandene Datenwege mehrfach genutzt werden. Die Ressourcen-Bindung wird in der Art verändert, daß einer Operation nicht mehr die erste freie Instanz eines ausgewählten Bausteintyps zugewiesen wird. Zuerst erfolgt für Operation j der Versuch einer Zuweisung zu Instanz $k = HW(j)$. Falls dieses nicht möglich ist, da die entsprechende Instanz bereits benutzt wird oder nicht ausgewählt werden darf, wird eine Zufallsauswahl zwischen den noch zur Auswahl stehenden Instanzen des Bausteintyps getroffen. Im Verlauf des genetischen Algorithmus setzen sich dann gute Zuweisungen zu Instanzen durch.

In Abbildung 3.21 ist ein Datenflußgraph mit den vier Operationen j_1 bis j_4 angegeben. Anhand dieses Beispiels wird im folgenden gezeigt, daß mit der bisherigen Vorgehensweise bei der Zuordnung der Operationen zu Instanzen nicht immer eine optimale Verdrahtung der Instanzen erreicht werden kann. Aus dem Datenflußgraphen können die Zuweisungen der Operationen j_1 bis j_4 zu Kontrollschritten nach einem Chromosomendurchlauf abgelesen werden. Es wird angenommen, daß alle Operationen an Instanzen vom Bausteintyp m gebunden werden. Zur Ausführung der Operationen werden zwei Instanzen benötigt, da die Operationen j_2 und j_3 in Kontrollschritt 2 parallel ausgeführt werden.

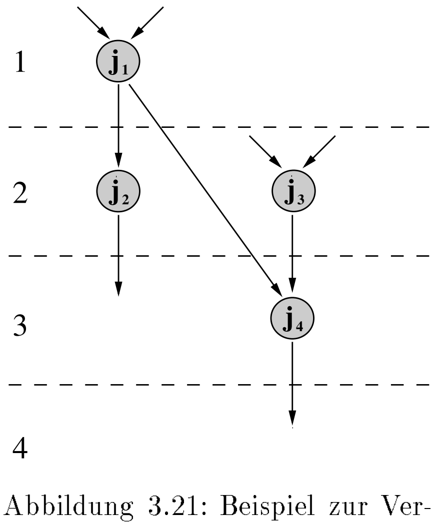


Abbildung 3.21: Beispiel zur Verbindungsminimierung

In dem gegebenen Datenflußgraphen sind die drei Vorrangsvorschriften $j_1 \prec j_2$, $j_1 \prec j_4$ und $j_3 \prec j_4$ vorhanden, für die jeweils ein Datenweg in Form einer Verbindung benötigt wird. Die vier Operationen werden zunächst in der Reihenfolge j_1, j_2, j_3, j_4 entsprechend ihrer Startzeitpunkte den Instanzen zugeordnet.

Die Tabelle tab_{bind} in Abbildung 3.22 stellt das Ergebnis der Zuweisung der Operationen zu den

Instanzen dar, wenn jeweils die erste freie Instanz ausgewählt wird.

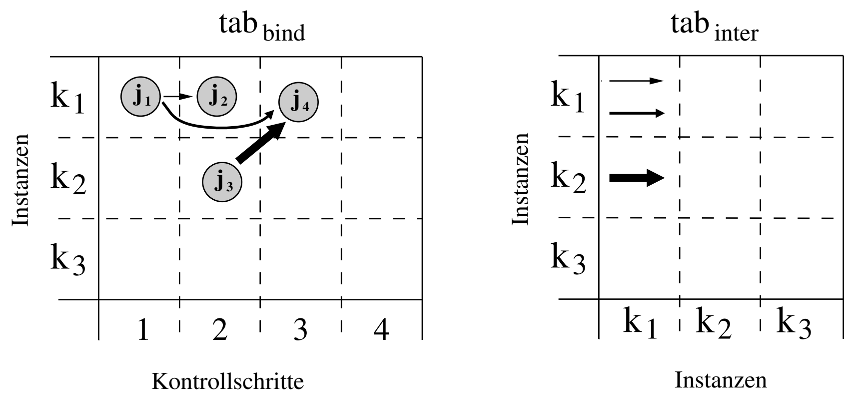


Abbildung 3.22: Zuordnung der Operationen zur ersten freien Instanz

Zur Verdeutlichung der im Datenflußgraphen vorhandenen Vorrangsvorschriften sind in der Tabelle zwischen den Operationen unterschiedlich breite Pfeile eingetragen. In Tabelle tab_{inter} werden die benötigten Verbindungen zwischen den Instanzen eingetragen und verwaltet. Für die Vorrangsvorschrift $j_1 \prec j_2$ ist hier eine Verbindung des Ausgangsports von Instanz k_1 zu einem Eingangsport von Instanz k_1 erforderlich, da beide Operationen auf dieser Instanz ausgeführt werden. Bei der dargestellten Zuweisung der Operationen werden insgesamt statt der maximal möglichen drei nur zwei Verbindungen benötigt, da die Verbindung w_{k_1,k_1} doppelt benutzt wird.

Da die zuvor beschriebene Zuweisung der Operationen zur Verbindungsminimierung nicht optimal ist, wird nun für jede Operation j versucht, diese zuerst der Instanz $k = HW(j)$ zuzuweisen. Das ist genau dann möglich, wenn diese Instanz des Bausteintyps benutzt werden darf und zu diesem Kontrollschritt noch nicht belegt ist. Falls keine Zuweisung der Operation j zu dieser Instanz möglich ist, wird mittels einer Zufallsauswahl eine freie Instanz bestimmt. Das folgende Beispiel (siehe Abbildung 3.23) zeigt, daß durch die Berücksichtigung der auf dem Chromosom kodierten Instanzen und einer teils zufälligen Zuweisung der Operationen zu den Instanzen eine optimale Ressourcen-Bindung möglich ist.

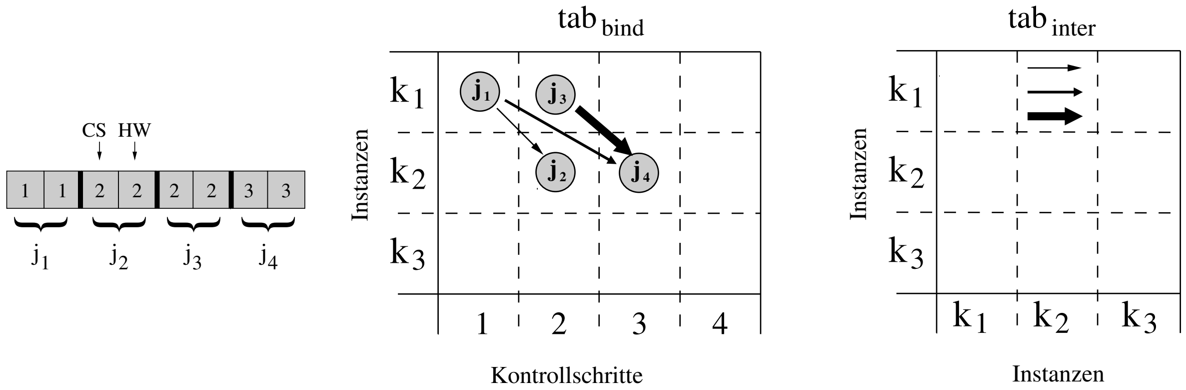


Abbildung 3.23: Veränderte Zuordnung der Operationen zu einer freien Instanz

Die Belegungen der Hardware-Gene auf dem Chromosom stellen bezüglich der Allokation gültige Ergebnisse dar. Jedoch ist zur Einhaltung der Baustein-Zuordnungsvorschrift noch eine Bindung der Operationen an die Instanzen erforderlich. Die Operationen werden wiederum in der Reihenfolge j_1, j_2, j_3, j_4 an die Instanzen gebunden. Der Instanz k_3 soll dabei keine Operation zugewiesen werden, da zwei Instanzen zur Ausführung ausreichen. Eine Zuordnung der Operationen j_1 und j_2 kann zu den Instanzen k_1 beziehungsweise k_2 entsprechend der Kodierung auf dem Chromosom vorgenommen werden. Bei dem Versuch, die Operation j_3 der Instanz k_2 zuzuordnen, wird festgestellt, daß diese bereits von der Operation j_2 belegt ist. Stattdessen muß der Operation j_3 die letzte freie benutzbare Instanz k_1 zugewiesen werden. Die Zuweisung der Operation j_4 zu der Instanz k_2 wird probabilistisch durchgeführt, da die Instanz $HW(j_4) = k_3$ nicht benutzt werden darf. In der Tabelle tab_{inter} in Abbildung 3.23 ist zu erkennen, daß nur noch eine Verbindung w_{k_1,k_2} zwischen den Instanzen k_1 und k_2 benötigt wird.

3.5 Realisierung des genetischen Algorithmus

Nachdem in den vorherigen Abschnitten beschrieben wurde, in welcher Weise die Nebenbedingungen bei der Mikroarchitektur-Synthese für ein Individuum berücksichtigt werden, beschäftigt sich dieser Abschnitt mit dem spezielleren Ablauf des genetischen Algorithmus.

Abbildung 3.24 gibt den bereits in Kapitel 2 vorgestellten allgemeinen Ablauf eines genetischen Algorithmus wieder. Der hier vorgestellte richtet sich im wesentlichen nach diesem Ablauf.
Dieser genetische Algorithmus terminiert mit Erreichen einer bestimmten Anzahl num_{gen} von Generationen.
In den folgenden Abschnitten wird die Durchführung der einzelnen Schritte für den entwickelten genetischen Algorithmus genauer beschrieben.

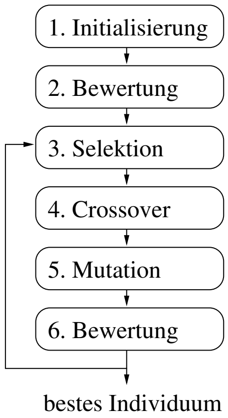


Abbildung 3.24: Ablauf des GAs

3.5.1 Initialisierung

Bei der Zuweisung einer Anfangsbelegung der Gene für die Individuen gilt es im wesentlichen zwei Ziele zu erreichen:

1. Erzeugung guter Individuen, um sich dem Optimum bereits zu Beginn des Optimierungsprozesses so gut wie möglich anzunähern.
2. Erzeugung möglichst unterschiedlicher Individuen, um einer vorzeitigen Konvergenz des Algorithmus in einem lokalen Optimum vorzubeugen.

Zur Realisierung dieser Ziele ist es erforderlich, daß für jedes Individuum der Population ein Chromosomendurchlauf durchgeführt wird. Die Initialisierungsfunktion wird zur Verdeutlichung zunächst in Pseudocode mit den wesentlichen Unterfunktionsaufrufen dargestellt. Zu Beginn eines Chromosomendurchlaufs gelten für alle Operationen $j \in J$ eines Individuums die folgenden initialen Werte:

1. $\text{ASAP}_{dyn}(j) = \text{ASAP}(j)$ und $\text{ALAP}_{dyn}(j) = \text{ALAP}(j)$
2. $ch_{accu}(j) = 0$

Die Nummern hinter den Funktionsaufrufen stellen Verweise auf die im Anschluß angegebenen Erklärungen zu den entsprechenden Unterfunktionen dar.

Pseudocode für die Initialisierungsfunktion:

```
FOR individuum := 1 TO popsize DO                               – für jedes Individuum
  {
    WHILE (j ≤ j_max) DO                                        – für jede Operation
        {
          IF macro_flag(j) = 1 THEN
             j := Next_Operation_Init(j);                                           (1)
          IF timing_list(j) ≠ NULL THEN
             Correct_Bounds_Timing (j, timing_list(j));                              (2)
          i := Scheduling_Init (j);                                                  (3)
          k := Allocation_Init (i, j);                                               (4)
          Correct_Bounds (i, j, k, timing_list(j), succ_list(j));                    (5)
        }
    IF interconnection_minimization THEN
       Binding ();                                                                   (6)
  }
```

(1) *Next_Operation_Init* (j)

Wird bei einem Chromosomendurchlauf eine Makrooperation erkannt, muß der gegenseitige Ausschluß dieser Makrooperation und den dazugehörigen Einzeloperationen realisiert werden (siehe Abschnitt 3.4.5). Dazu wird an dieser Stelle entschieden, welche der beiden Alternativen ausgewählt wird. Durch bereits vorgenommene Kontrollschritt-Einschränkungen und unterschiedliche Ausführungszeiten der Makrooperation und der Einzeloperationen kann dar Fall eintreten, daß nur eine der Alternativen zu einer gültigen Lösung führt. Bei der Entscheidung wird deswegen berücksichtigt, ob eine der Alternativen eine gültige Kontrollschritt-Belegung erlaubt oder nicht. Die Überprüfung kann durch einen Vergleich der unteren und oberen dynamischen Kontrollschritt-Grenzen erfolgen. Wurde die untere dynamische Grenze einer Operation so weit hochgesetzt, daß sie größer als die obere ist, kann bei der Realisierung dieser Operation keine gültige Lösung mehr erreicht werden. Andernfalls wird bei Entscheidungsfreiheit zufällig eine der beiden Alternativen ausgewählt.

Falls die Einzeloperationen $j+1, \ldots, j+n$ realisiert werden sollen, wird im aktuellen Schleifendurchlauf die Makrooperation j übersprungen und die erste Einzeloperation $j+1$ behandelt. Im kommenden Schleifendurchlauf wird mit der nächsten Einzeloperation $j+2$ fortgefahren. Andererseits werden bei der Realisierung der Makrooperation j beim nächsten Schleifendurchlauf die folgenden n Einzeloperationen $(j+1), \ldots, (j+n)$ nicht behandelt.

(2) *Correct_Bounds_Timing* $(j, timing_{list}(j))$

In dieser Funktion werden zunächst Grenzeinschränkungen von Operation j aufgrund von Zeitvorgabevorschriften zu anderen noch nicht besuchten Operationen durchgeführt. Durch die Einschränkungen der Kontrollschritt-Grenzen $\mathrm{ASAP}_{dyn}(j)$ und $\mathrm{ALAP}_{dyn}(j)$ von Operation j werden diejenigen Belegungen des Kontrollschritt-Gens vermieden, die zwangsläufig zu einer ungültigen Lösung führen würden. Die vorzunehmenden Einschränkungen der dynamischen Grenzen sind in Abschnitt 3.4.6 auf den Seiten 31 ff beschrieben.

Bei einer Erhöhung von $\mathrm{ASAP}_{dyn}(j)$ wird der Akkumulator $ch_{accu}(j)$ von der Operation j auf 0 ns zurückgesetzt, damit bei Bedarf die verbliebene Restausführungszeit $t_{remain}(j, \mathrm{ASAP}_{dyn}(j))$ von Operation j korrekt berechnet werden kann.

(3) *Scheduling_Init* (j)

Die Aufgabe dieser Funktion besteht in der Zuweisung der aktuellen Operation j zu einem Kontrollschritt i aus $\{\mathrm{ASAP}_{dyn}(j), \ldots, \mathrm{ALAP}_{dyn}(j)\}$. Es wird zuvor sichergestellt, daß zu Kontrollschritt $i = \mathrm{ASAP}_{dyn}(j)$ eine genügend schnelle Instanz eines geeigneten Bausteintyps vorhanden ist, die eine Ausführung der Operation j zu diesem Zeitpunkt innerhalb der Restausführungszeit $t_{remain}(j, \mathrm{ASAP}_{dyn}(j))$ ermöglicht.

Die Auswahl eines Ausführungszeitpunktes für die Operationen wird nach verschiedenen Strategien vorgenommen, um möglichst unterschiedliche Individuen innerhalb einer Population zu erhalten und somit eine vorzeitige Konvergenz zu vermeiden.

a) Die Operation j wird dem frühestmöglichen Startzeitpunkt $i = \mathrm{ASAP}_{dyn}(j)$ zugewiesen (ASAP-Scheduling).

b) Die Operation j wird zufällig einem der beiden Kontrollschritte

$$\left\lfloor \frac{\mathrm{ASAP}_{dyn}(j)+\mathrm{ALAP}_{dyn}(j)}{2} \right\rfloor \quad \text{oder} \quad \left\lceil \frac{\mathrm{ASAP}_{dyn}(j)+\mathrm{ALAP}_{dyn}(j)}{2} \right\rceil$$

zugewiesen.

c) Die Operation j wird dem spätestmöglichen Startzeitpunkt $i = \mathrm{ALAP}_{dyn}(j)$ zugewiesen (ALAP-Scheduling).

d) Die Zuordnung von Operation j zu einem Kontrollschritt i wird von der Position auf dem Chromosom abhängig gemacht. Die ersten Operationen werden dabei dem frühestmöglichen dynamischen Ausführungszeitpunkt zugewiesen. Im Chromosom weiter hinten stehende Operationen erhalten immer mehr Möglichkeiten bei der Auswahl eines Kontrollschrittes. Für Operationen mit den höchsten Ordnungsnummern wird letztlich auch die Auswahl des spätestmöglichen Ausführungszeitpunktes gestattet.

Aufgrund bestehender Datenabhängigkeiten zwischen den Operationen wirkt sich die Wahl eines späten Ausführungszeitpunktes für eine Operation mit kleiner Ordnungsnummer stark auf andere Operationen aus. Auf die Durchführung einer vollständigen Zufallsauswahl wird deswegen verzichtet.
Der ausgewählte Kontrollschritt i wird bei der Initialisierung nicht weiter verändert und kann deswegen im Chromosom als Allel für das Gen $CS(j)$ gesetzt werden.

(4) *Allocation_Init* (i, j)

Die Aufgabe dieser Funktion besteht in der Zuweisung eines geeigneten Bausteintyps zu Operation j und realisiert die Einhaltung der Baustein-Zuordnungsvorschrift (siehe Abschnitt 3.4.2).

Aus der Menge der Bausteintypen, auf denen die Operation j ausgeführt werden kann, wird probabilistisch ein Bausteintyp m ausgewählt. Falls durch diese Auswahl eine zusätzliche Instanz von Bausteintyp m benötigt wird, kann mit der Wahrscheinlichkeit $prob_m \in [0,1]$

der Bausteintyp gewechselt werden. Ein Wechsel auf einen anderen geeigneten Bausteintypen m' wird allerdings nur dann durchgeführt, wenn bei dessen Auswahl zu diesem Zeitpunkt keine zusätzliche Instanz benötigt wird. Mit Hilfe der Tabelle tab_{alloc} kann jeweils die Anzahl der benötigten Instanzen bestimmt werden. In Abbildung 3.25 wird anhand der Tabelle tab_{alloc} diese Vorgehensweise verdeutlicht.

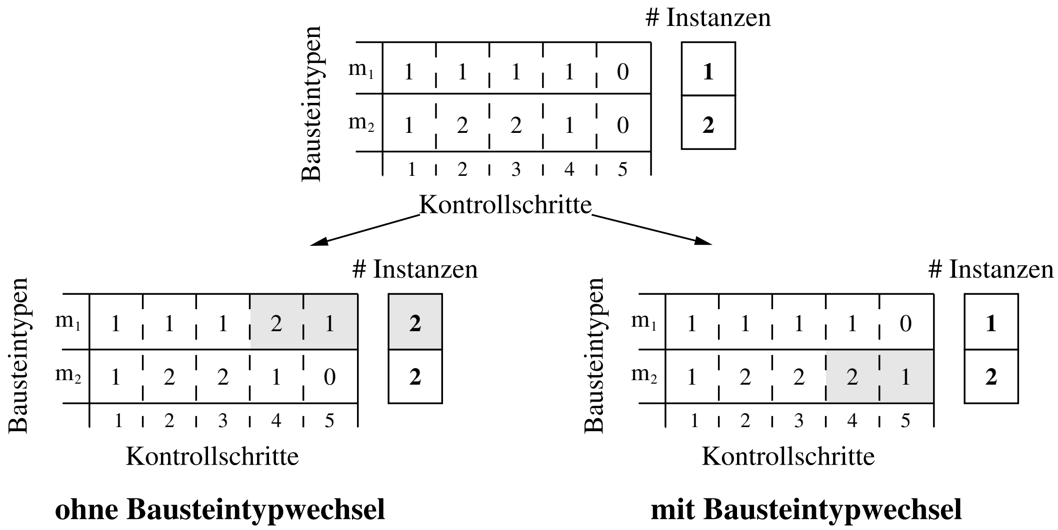


Abbildung 3.25: Tabelle tab_{alloc} mit und ohne Wechsel des Bausteintyps

Die oberste Tabelle zeigt den Zustand der Eintragungen in die Tabelle tab_{alloc} vor der Zuweisung von Operation j zu einem Bausteintyp. Es wird angenommen, daß die Operation j den Bausteintypen m_1 und m_2 zugewiesen werden kann. Weiterhin wurde bereits als Ausführungszeitpunkt der Kontrollschritt 4 und als vorläufiger Bausteintyp m_1 bestimmt. Die Latenzzeiten der Bausteintypen bei der Ausführung von Operation j betragen jeweils 2 Kontrollschritte. Daraus folgt, daß die Operation j in den Kontrollschritten 4 *und* 5 in der Tabelle tab_{alloc} die Einträge in einer der Zeilen m_1 oder m_2 erhöhen wird.
Die rechte untere Tabelle gibt den Endzustand wieder, falls der Bausteintyp von m_1 auf m_2 gewechselt werden darf. Anders als bei der Zuweisung zum Bausteintyp m_1 erspart ein Wechsel des Bausteintyps auf m_2 zu diesem Zeitpunkt des Chromosomendurchlaufs die Benutzung einer zusätzlichen Instanz.

Im Chromosom wird das Gen $HW(j)$ von Operation j mit einer Instanz des ermittelten Bausteintyps belegt und für dieses Individuum während der Initialisierungsfunktion nicht mehr verändert.

(5) *Correct_Bounds* $(i, j, k, timing_{list}(j), succ_{list}(j))$

In dieser Funktion werden die dynamischen Kontrollschritt-Grenzen und Akkumulatoren der Operationen korrigiert, die in direkter Beziehung mit der aktuellen Operation j stehen. Dabei werden die Vorrangsvorschriften und Zeitvorgabevorschriften zu Operationen behandelt, die eine höhere Ordnungsnummer besitzen und somit während des Chromosomendurchlaufs noch nicht besucht wurden.
Die Zeitvorgabevorschriften zwischen der aktuellen Operation j und weiteren Operationen j' werden durch die in Abschnitt 3.4.6 auf den Seiten 35 ff vorgestellten Einschränkungen der dynamischen Grenzen von Operation j' eingehalten.

Zur Berücksichtigung von Chaining werden an dieser Stelle zunächst die Akkumulatoren $ch_{accu}(j')$ aller Nachfolgeoperationen j' mit Formel (3.2) oder (3.3) geändert. Danach erfolgt eine Änderung von $\mathrm{ASAP}_{dyn}(j')$ mit Formel (3.1) zur Einhaltung der Vorrangsvorschriften.

(6) *Binding* ()

Für den Fall, daß eine Verbindungsminimierung durchgeführt werden soll, müssen alle Operationen an konkrete Instanzen gebunden werden. Die Zuweisung von Operationen zu den Instanzen erfolgt durch Zuweisung einer probabilistisch ausgewählten Instanz (siehe Abschnitt 3.4.7).

3.5.2 Bewertung

Die Bewertung der Individuen einer Population dient zur Differenzierung der Individuen und ist Voraussetzung für die Durchführung des Selektionsschrittes. Grundsätzlich stellen Individuen, die geringe Kosten verursachen, bessere Lösungen dar, als solche mit höheren Kosten. Allerdings können auch aufgrund der Komplexität der Problemstellung einzelne Individuen gegen Nebenbedingungen verstoßen und damit ungültige Lösungen darstellen. Diese Verstöße werden bei der Bewertung der Individuen berücksichtigt, indem die in Abschnitt 3.3 beschriebene Kostenfunktion um einen „Bestrafungsterm“ erweitert wird. Die Höhe der Bestrafung richtet sich dabei nach dem Ausmaß des Fehlers. Mit jedem Individuum ind wird deswegen ein Strafwert $penalty(ind)$ verbunden, der den Wert 0 annimmt, wenn das Individuum eine gültige Lösung darstellt.

Fehler werden beim Scheduling während eines Chromosomendurchlaufs erkannt, falls einer Operation j kein gültiger Kontrollschritt zugewiesen werden kann. Das ist genau dann der Fall, wenn für die Operation j $\mathrm{ASAP}_{dyn}(j) > \mathrm{ALAP}_{dyn}(j)$ gilt. Um das Ausmaß des Fehlers zu berücksichtigen, wird in diesem Fall der Strafwert $penalty(ind)$ des betrachteten Individuums ind um die Differenz der beiden dynamischen Grenzen erhöht. Offensichtlich nähern Individuen, die gültige Lösungen darstellen und hohe Kosten verursachen, das Optimum besser an, als Individuen mit geringen Kosten, die gegen Nebenbedingungen verstoßen. Aus diesem Grund geht der mit $penalty(ind)$ identifizierte Strafwert mit einem Gewicht von w_{pen} in die Berechnung der Kosten für jedes Individuum ein. Eine sinnvolle Gewichtung w_{pen} ergibt sich durch die maximal möglichen Kosten $cost_{max}$.

Bei der Spezifikation eines Entwurfs besteht zwecks Kostenreduzierung die Möglichkeit, die erlaubte Gesamtausführungszeit des Entwurfs zu erhöhen. In einigen Fällen wird die vollständige Ausnutzung der erlaubten Ausführungszeit allerdings nicht erforderlich sein. Bei zwei Individuen mit denselben Kosten, soll deswegen das Individuum bevorzugt werden, das eine geringere Ausführungszeit des Entwurfs ermöglicht. Dies führt zu einer zusätzlichen Erweiterung der Kostenfunktion, indem die Anzahl der benötigten Kontrollschritte num_{cs} eines Entwurfs mit dem Gewicht w_{cs} integriert wird.

Jedem Individuum ind werden also die Kosten mittels der folgenden erweiterten Kostenfunktion zugewiesen:

$$\mathrm{Instanzenkosten}(ind) + \mathrm{Verbindungskosten}(ind) + w_{pen} * penalty(ind) + w_{cs} * num_{cs}(ind)$$

Danach werden die Individuen nach Kosten sortiert und Rängen zugewiesen, wodurch Individuen mit niedrigen Kosten einen kleinen Rang erhalten. Die aufgrund der Ränge berechneten Fitneßwerte stellen dann die Grundlage für die Selektion dar.

3.5.3 Selektion

Vor der eigentlichen Selektion wird eine Elite-Selektion durchgeführt, indem die besten $num_{repl} \in \{1, \ldots, popsize\}$ Individuen ohne Veränderung in die nächste Generation übernommen werden. Dadurch wird garantiert, daß immer die besten num_{repl} Individuen in der nachfolgenden Generation vorhanden sind. Die Selektion hat nun die Aufgabe, auf der Basis der Fitneßwerte der Individuen einer Population die Individuen auszuwählen, die ihre Gene in die nächste Generation vererben sollen. Zur Durchführung der Selektion wird der in Abschnitt 2.3 beschriebene Auswahlalgorithmus SUS (Stochastic Universal Sampling) verwendet. Die im Genpool vorhandenen Individuen liefern das Genmaterial zur Durchführung des Crossover, das im folgenden Abschnitt beschrieben wird.

3.5.4 Crossover

Das Crossover hat die Aufgabe, aus dem in der Population vorhandenem Genmaterial neue Individuen zu erzeugen, die jedes für sich wiederum eine Lösung des Problems darstellen. Dazu werden die Gene zweier Eltern derart kombiniert, daß zwei Nachkommen entstehen, die teilweise aus Genen des einen und des anderen Elter bestehen. Ein Crossover wird zwischen zwei selektierten Eltern mit einer Wahrscheinlichkeit von $prob_{cross} \in [0, 1]$ durchgeführt. Als Crossover-Variante wird das Uniform Crossover realisiert, bei dem jedes Gen zweier Eltern mit derselben Wahrscheinlichkeit $(1 - prob_{uni}) \in [0, 1]$ ausgetauscht wird. Falls $prob_{uni} = 0.6$ gilt, dann wird also mit einer Wahrscheinlichkeit von 40 % ein bestimmtes Gen ausgetauscht. Für den Fall, daß die Operation des einen Elter in Hardware realisiert werden soll und die des anderen Elter nicht, findet zur Vermeidung von Inkonsistenzen bezüglich des gegenseitigen Ausschlusses einer Makrooperation und den dazugehörigen n Einzeloperationen kein Austausch von Genen statt. Zur Verdeutlichung des Uniform Crossover und der Auswirkungen des Crossover auf die erzeugten Nachkommen wird die Durchführung an einem Beispiel erklärt. In Abbildung 3.26 sind die Datenflußgraphen zweier Eltern p_1 und p_2 mit den fünf Operationen j_1 bis j_5 dargestellt.

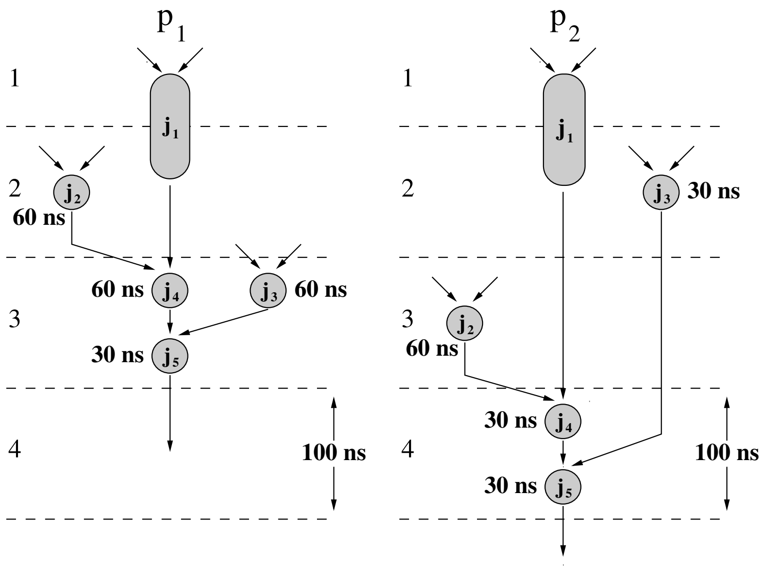


Abbildung 3.26: Datenflußgraphen zweier Eltern p_1 und p_2

Eine Instanz von Bausteintyp m_1 benötigt 2 Kontrollschritte, um die Operation j_1 auszuführen. Die Operationen j_2 bis j_5 können auf Instanzen von Bausteintyp m_2 mit einer Ausführungszeit von 30 ns und auf Instanzen von Bausteintyp m_3 innerhalb von 60 ns ausgeführt werden. Bei einer Zykluszeit von 100 ns soll Chaining zwischen den Operationen j_2 bis j_5 berücksichtigt werden. Offensichtlich stellen die Datenflußgraphen der beiden Eltern p_1 und p_2 gültige Lösungen dar. Die Chromosomen der beiden Eltern p_1 und p_2 sind in Abbildung 3.27 dargestellt. Zugunsten einer besseren Übersicht ist anstelle einer Instanz der entsprechende Bausteintyp angegeben.

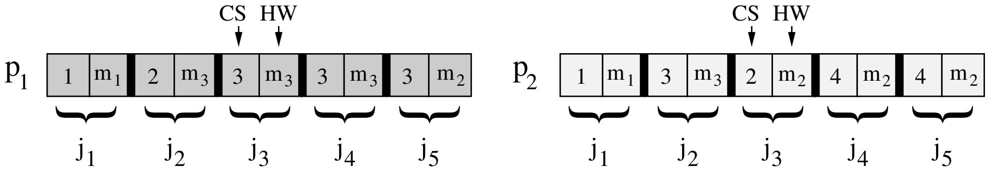


Abbildung 3.27: Chromosomen der Eltern p_1 und p_2

Bei der Durchführung des Crossover werden das Kontrollschritt-Gen und das Hardware-Gen einer Operation gemeinsam ausgetauscht. In diesem Beispiel sollen die beiden Gene der Operation j_4 ausgetauscht werden. Daraus ergeben sich die in Abbildung 3.28 dargestellten Chromosomen der Nachkommen c_1 und c_2.

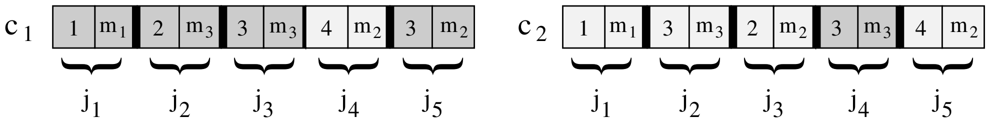


Abbildung 3.28: Chromosomen der Nachkommen c_1 und c_2

Es ist zu erkennen, daß beide Gene von Operation j_4 der Eltern ausgetauscht worden sind. Daraus resultieren die in der Abbildung 3.29 abgebildeten neuen Datenflußgraphen der Nachkommen c_1 und c_2.

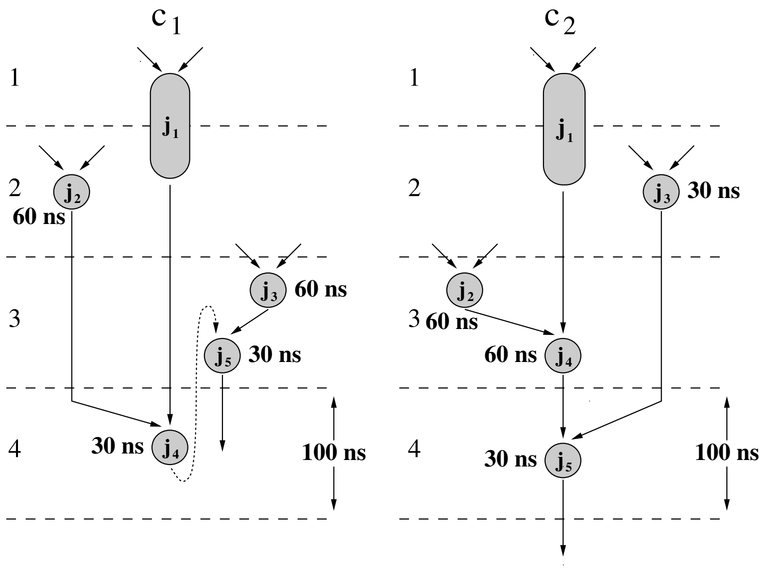


Abbildung 3.29: Datenflußgraphen der Nachkommen c_1 und c_2

Anhand der dargestellten Datenflußgraphen läßt sich erkennen, daß mit der Durchführung des Crossover ungültige Individuen entstehen können. In diesem Beispiel treten zweierlei Verletzungen der Nebenbedingungen auf. Im Datenflußgraphen von Nachkommen c_1 ist zu erkennen, daß durch den Austausch der Gene von Operation j_4 die Vorrangsvorschrift zwischen den Operationen j_4 und j_5 nicht mehr eingehalten wird (siehe gestrichelte Linie).
Ein anderes Problem läßt sich hingegen bei der Betrachtung des Datenflußraphen des Nachkommen c_2 erkennen. Durch den Austausch der Gene werden nun die beiden Operationen j_2 und j_4 in Kontrollschritt 3 ausgeführt und verkettet. Die kumulierte Ausführungszeit $60\ ns + 60\ ns = 120\ ns$ der Operationen führt zu einer Überschreitung der Zykluszeit t_{cs} von $100\ ns$, da beide Operationen auf Instanzen von Bausteintyp m_3 ausgeführt werden.

Damit keine ungültigen Individuen in die nächste Generation übernommen werden, bestehen an dieser Stelle zwei grundsätzliche Möglichkeiten zur weiteren Vorgehensweise.

1. Den Austausch der beiden Gene von Operation j_4 wieder rückgängig machen (und optional versuchen, die Gene einer anderen Operation auszutauschen).

2. Die entstandenen Individuen in der Art korrigieren, daß durch möglichst geringe Veränderungen der Gene wieder gültige Lösungen entstehen.

Die Durchführung der ersten Alternative kann sich bei genauerer Betrachtung als äußerst ungünstig erweisen, da in einem stark eingeschränkten Lösungsraum viele Austauschaktionen wieder rückgängig gemacht werden müssen. Dadurch wird die Effektivität des Crossover eingeschränkt und zusätzlich viel Rechenzeit vergeudet. Für die Entscheidung, ob der erzeugte Nachkomme alle Nebenbedingungen erfüllt, reicht die Überprüfung lokaler direkter Beziehungen zu anderen Operationen nicht aus, da bei der Berücksichtigung von Chaining die vollständige Operationen-Kette betroffen ist und nicht nur die direkten Vor- und Nachfolgeoperationen.
Die zweite der oben erwähnten Alternativen läßt sich mittels eines Chromosomendurchlaufs realisieren, mit dem für ein Individuum effizient bestimmt werden kann, ob alle Nebenbedingungen von allen Operationen eingehalten werden. Die Belegungen der Gene können dabei als feststehende Werte angenommen werden, auf deren Basis die Einschränkungen zu anderen Operationen durchgeführt werden. Falls ein Verstoß einer Operation festgestellt wird, besteht nun die Möglichkeit, den vorhandenen Wert auf dem Chromosom durch eine möglichst minimale Änderung gültig zu setzen. Der Chromsomendurchlauf kann fortgesetzt und ein gültiges Individuum erzeugt werden, so daß keine Rechenzeit unnötig vergeudet wird. Das entspricht auch der Vorgehensweise in diesem genetischen Algorithmus.

In dem aktuellen Beispiel führt diese Vorgehensweise zu den in der Abbildung 3.30 dargestellten, korrigierten Datenflußgraphen der Nachkommen c_1 und c_2.

Für die Korrektur des Datenflußgraphen von Nachkommen c_1 sorgt während des Chromosomendurchlaufs die Betrachtung der Operation j_4. Aufgrund der bestehenden Vorrangsvorschrift $j_4 \prec j_5$ wird der $\text{ASAP}_{dyn}(j_5)$-Wert von der Operation j_5 mittels Formel (3.1) von 3 auf 4 hochgesetzt. Dadurch kann die aktuelle Belegung des Kontrollschritt-Gens von $CS(j_5) = 3$ als nicht realisierbar erkannt und durch Kontrollschritt 4 ersetzt werden. Zur Korrektur des Datenflußgraphen des Nachkommen c_2 reicht es aus, für die Operation j_4 einen schnelleren Bausteintyp auszuwählen. Dazu wird bei der Betrachtung von Operation j_2 der Akkumulator $ch_{accu}(j_4)$ von Operation j_4 mit Formel (3.2) auf $60\ ns$ hochgesetzt, so daß als einzige Ausführungsmöglichkeit für Operation j_4 eine Instanz von Bausteintyp m_2 in Frage kommt, um die Einhaltung der Zykluszeit von $100\ ns$ sicherzustellen.

Mit der Durchführung eines Chromosomendurchlaufs für jedes Individuum nach einem Crossover ergeben sich ebenfalls große Vorteile bei der im folgenden Abschnitt beschriebenen Realisierung der Mutation.

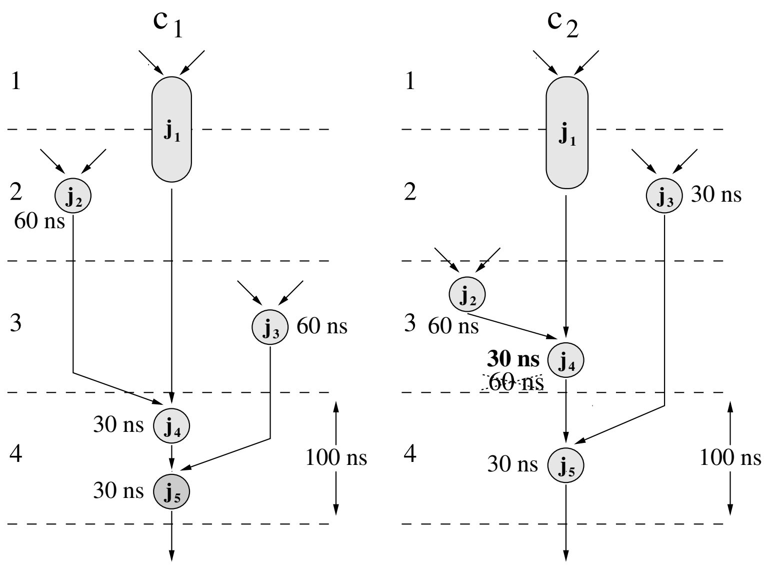


Abbildung 3.30: Datenflußgraphen der Nachkommen c_1 und c_2 nach der Korrektur

3.5.5 Mutation

In einem genetischen Algorithmus hat die Mutation die Aufgabe, neues Genmaterial zu erzeugen oder im Verlauf des Evolutionsprozesses verlorengegangenes Genmaterial wiederzugewinnen. Eine sinnvolle Mutation ist nur möglich, wenn Kenntnisse über den Bereich gültiger neuer Belegungen vorliegen. Da nach einem Crossover bereits ein Chromosomendurchlauf für jeden erzeugten Nachkommen durchgeführt wird, bietet sich eine Kombination von Korrektur und Mutation an. Die Belegung des Kontrollschritt-Gens $CS(j)$ von Operation j wird mit einer Wahrscheinlichkeit von $prob_{cs} \in [0, 1]$ und die Belegung des Hardware-Gens $HW(j)$ von Operation j mit einer Wahrscheinlichkeit von $prob_{hw} \in [0, 1]$ mutiert.

Der Ablauf entspricht im wesentlichen dem der Initialisierungsfunktion. Unterschiede in den Funktionen ergeben sich an Stellen, bei denen aktuelle Genbelegungen berücksichtigt werden können. Ausgedrückt werden die Unterschiede durch die Zusätze *_Init* beziehungsweise *_Mutat*. Gegenüber der Initialisierungsfunktion ergeben sich für drei Unterfunktionen Änderungen:

(1) *Next_Operation_Mutat (j)*

Mit dieser Funktion wird analog zur Initialisierung der gegenseitige Ausschluß einer Makrooperation j und den dazugehörigen n Einzeloperationen $(j + 1), \ldots, (j + n)$ realisiert. Anders als bei der Initialisierung werden an dieser Stelle aktuelle Belegungen der Gene mitberücksichtigt. Dabei ist es hilfreich, daß bei der Durchführung des Crossover der Austausch von Genen bestimmter Operationen ausgeschlossen wird, um an dieser Stelle Entscheidungskonflikte zu vermeiden. Hierdurch wird verhindert, daß sowohl eine Makrooperation als auch eine oder mehrere dazugehörige Einzeloperationen gleichzeitig in Hardware realisiert werden.

Analog zur Initialisierungsphase werden die vorhandenen Ausführungsalternativen der Makrooperation und der Einzeloperationen geprüft. Dabei existieren drei grundsätzliche Möglichkeiten:

1. beide Alternativen führen zu ungültigen Individuen
   $\Rightarrow$ die aktuelle Ausführungsvariante wird zunächst beibehalten

2. es ist nur eine Alternative durchführbar
   ⇒ es wird die entsprechend durchführbare Variante ausgewählt

3. es sind beide Alternativen ausführbar
   ⇒ die aktuelle Ausführungsvariante wird zunächst beibehalten

Anschließend kann als Besonderheit bei der ersten und dritten Möglichkeit eine Mutation mit der Wahrscheinlichkeit $prob_{mac} \in [0,1]$ durchgeführt werden, indem die nicht der aktuellen Belegung entsprechende Alternative ausgewählt wird.

(3) *Scheduling_Mutat (j)*

Mit der Wahrscheinlichkeit $prob_{cs}$ wird für einen Kontrollschritt eine Mutation aufgrund der aktuellen Belegung durchgeführt. Dazu wird zunächst die Richtung bestimmt, in die eine Veränderung des aktuellen Kontrollschrittes erfolgen soll. Um eine Tendenz zu späten Ausführungszeitpunkten zu vermeiden, wird deswegen mit der Wahrscheinlichkeit $prob_{dir}$ diese Entscheidung beeinflußt. Für $prob_{dir} = 0.6$ wird dann zu 60 % ein früherer Ausführungszeitpunkt als der aktuelle bestimmt. Die Auswahl des neuen Kontrollschrittes erfolgt probabilistisch, wobei sichergestellt wird, daß ein Kontrollschritt ausgewählt wird, der zu einer gültigen Lösung führt. Wird keine Mutation durchgeführt, muß überprüft werden, ob der aktuell auf dem Chromsom kodierte Kontrollschritt $CS(j)$ von Operation j einem Wert aus $\{\text{ASAP}_{dyn}(j), \ldots, \text{ALAP}_{dyn}(j)\}$ entspricht. Falls nicht, wird zur Vermeidung ungültiger Lösungen ein neuer Kontrollschritt für $CS(j)$ bestimmt, wobei immer der Kontrollschritt mit der geringsten Veränderung zu vorher ausgewählt wird.

(4) *Allocation_Mutat (i, j)*

Falls die aktuelle Zuweisung zu einem Bausteintyp nicht zu einer gültigen Lösung führt oder mit der Wahrscheinlichkeit $prob_{hw}$ eine aktuelle Belegung des Hardware-Gens verändert werden soll, wird aus allen vorhandenen gültigen Belegungen ein neuer Bausteintyp in Form einer Instanz zufällig ausgewählt.
Wird durch die aktuelle Auswahl eine zusätzliche Instanz von Bausteintyp m benötigt, kann analog zur Funktion *Allocation_Init* mit der Wahrscheinlichkeit $prob_m$ der ausgewählte Bausteintyp m gewechselt werden.

In den letzten Abschnitten wurde eine Methode entwickelt, mit der die Mikroarchitektur-Synthese unter Berücksichtigung eines großen Funktionsumfangs mit einem genetischen Algorithmus durchgeführt werden kann. Im folgenden Abschnitt wird das im Rahmen dieser Diplomarbeit entwickelte und implementierte Gesamtsystem beschrieben.

3.6 Das Gesamtsystem

Zur Durchführung der Synthese ist eine Anbindung an ein Synthese-System erforderlich, das die Mikroarchitektur-Synthese auf der Basis der ganzzahlig linearen Programmierung realisiert. Mit dieser Anbindung wird der eigentliche Synthese-Kern durch das entwickelte Gesamtsystem mit integriertem genetischen Algorithmus ersetzt. Ein Überblick über das Gesamtsystem in Verbindung mit dem OSCAR-System wird in Abbildung 3.31 gegeben.

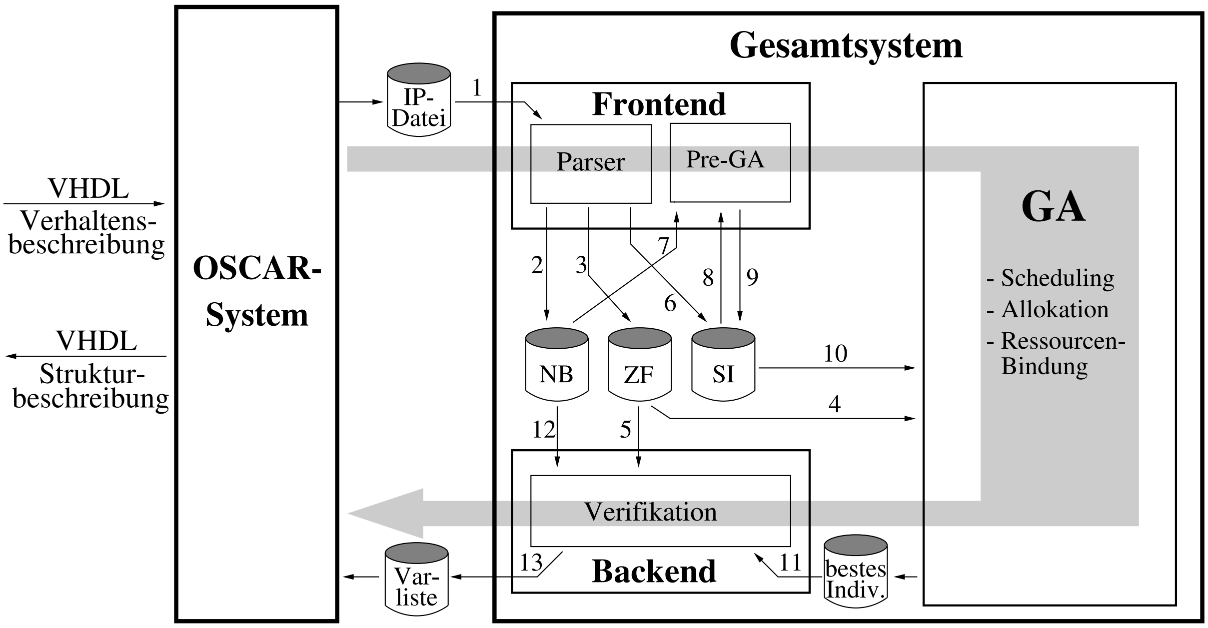


Abbildung 3.31: Das Gesamtsystem in Verbindung mit dem OSCAR-System

In einem Preprocessing-Schritt liest das OSCAR-System eine gegebene VHDL-Verhaltensbeschreibung ein. Vor der Generierung der IP-Datei ist es dem Designer möglich, mittels einer komfortablen graphischen Benutzeroberfläche [Sei95] Entwurfs-Spezifikationen anzugeben. Dies betrifft zum Beispiel die Erhöhung der Anzahl erlaubter Kontrollschritte, Durchführung von Chaining und die Angabe von Zeitvorgabevorschriften. Weiterhin besteht unter anderem die Möglichkeit auf existierende Bausteinbibliotheken zurückzugreifen. Die vom OSCAR-System generierte IP-Datei dient dann als Eingabe für das Gesamtsystem. Dieses wird, wie in Abbildung 3.31 angedeutet, in die drei folgenden Bereiche unterteilt:

1. Das *Frontend* übernimmt die Vorverarbeitung der IP-Datei für den genetischen Algorithmus. Dieser Bereich untergliedert sich in die beiden Module Parser und Pre-GA.

    - Der *Parser* liest eine vom OSCAR-System erzeugte IP-Datei (Markierung 1) ein und legt die darin enthaltene Zielfunktion (ZF) und die Nebenbedingungen (NB) in einer internen Datenstruktur ab (Markierungen 2 und 3). Die IP-Zielfunktion dient als Grundlage für die Bewertung der Individuen des genetischen Algorithmus und der abschließenden Berechnung des Zielfunktionswertes der ermittelten besten Lösung (Markierungen 4 und 5).
    Weiterhin werden bereits beim Einlesen der IP-Datei durch den Parser wichtige Synthese-Informationen (SI) für den Ablauf des genetischen Algorithmus extrahiert (Markierung 6). Dazu gehört zum Beispiel die Anzahl der Operationen, die erlaubte Anzahl von Kontrollschritten sowie die Anzahl der Bausteintypen und Instanzen.
    - Das Modul *Pre-GA* stellt die unmittelbare Vorstufe des genetischen Algorithmus dar. Die für den Ablauf des genetischen Algorithmus erforderlichen restlichen Informationen werden hier mit Hilfe der Nebenbedingungen und der bereits vorhandenen Synthese-Informationen ermittelt (Markierungen 7, 8 und 9). Das betrifft unter anderem die Erzeugung einer Liste mit allen Vorrangs- und Zeitvorgabevorschriften für

jede Operation sowie die Ausführungszeiten und Latenzzeiten der einzelnen Bausteine.

Mit dem Abschluß dieser Phase stehen alle zum Ablauf des genetischen Algorithmus erforderlichen Informationen zur Verfügung.

2. Auf der Basis der ermittelten Daten über den zu optimierenden System-Entwurf führt der *genetische Algorithmus* die Optimierung der Zielfunktion unter den gegebenen Nebenbedingungen durch (Markierungen 4 und 10). Im wesentlichen erfolgt die Steuerung des genetischen Algorithmus über die im vorigen Abschnitt eingeführten Parameter. Dies sind insbesondere Populationsgröße, Mutationsraten für Kontrollschritte und Hardware-Bausteine sowie die Crossover-Wahrscheinlichkeit. Für alle Parameter sind Default-Werte vorgegeben. Änderungen der Parametereinstellungen durch den Designer können auf die in Anhang A beschriebene Art durchgeführt werden.

3. Das *Backend* hat die Aufgabe, die vom genetischen Algorithmus gefundene Lösung zu verifizieren und in Form einer Variablenliste an das OSCAR-System weiterzugeben (Markierung 13). Die Überprüfung der ermittelten Lösung wird anhand der IP-Nebenbedingungen durchgeführt (Markierung 12). Dazu werden für die in der IP-Datei vorhandenen Variablen konkrete Werte eingesetzt, die sich aus der berechneten Lösung ergeben (Markierung 11). Die ermittelte Lösung ist genau dann korrekt, falls gegen keine in der IP-Datei angegebene Nebenbedingung verstoßen wird. Damit wird im Sinne der *Correctness-by-Construction* ein wichtiger Aspekt beim Hardware-Entwurf berücksichtigt. Das Ergebnis wird zum Schluß mit dem Zielfunktionsergebnis in Form einer Variablenliste an das OSCAR-System zurückgegeben.

Mit der Integration des genetischen Algorithmus in das Gesamtsystem besteht nun die Möglichkeit vom OSCAR-System generierte IP-Dateien einzulesen und eine Optimierung der dort angegebenen Zielfunktion unter Berücksichtigung der Nebenbedingungen durchzuführen. Zur Beurteilung der Qualität dieses Verfahrens werden im folgenden Kapitel Testergebnisse für einige Benchmarks vorgestellt.

Kapitel 4

Testergebnisse

In diesem Kapitel werden Testergebnisse mit unterschiedlichen Design-Bedingungen vorgestellt. Als Benchmarks dienen hier der Elliptical-Wave-Filter (EWF) [KWK85], die Fast-Fourier-Transformation (FFT) [Ach95] und die Fast-Discret-Cosinus-Transformation (FDCT) [MD90]. Es werden unter anderem Untersuchungen bezüglich einer Erhöhung der Anzahl zulässiger Kontrollschritte, Chaining, Zeitvorgabevorschriften und Verbindungsminimierung unter Benutzung unterschiedlicher Bausteinbibliotheken durchgeführt. Bei den Bausteinbibliotheken werden einerseits mehrzyklische Bausteintypen, Komplexbausteine (hier: MACs) und Pipeline-Bausteine verwendet. Zusätzlich besteht für die Zuweisung von Additionen und Multiplikationen die Auswahlmöglichkeit zwischen verschiedenen Bausteintypen, die zwar dieselbe Funktionalität aufweisen, sich allerdings in bezug auf Kosten und Ausführungszeit unterscheiden.

Die Tests erfolgen mit zwei genetischen Algorithmen (GA1 und GA2), die bis auf die Verwendung einer anderen Sequenz von Zufallszahlen identisch sind. Dazu wird der Zufallszahlengenerator von GA1 und GA2 mit unterschiedlichen Werten (*seed*) initialisiert[1]. Die von den beiden genetischen Algorithmen berechneten Ergebnisse basieren auf denselben in Anhang A angegebenen internen Parametereinstellungen. Hiermit soll gezeigt werden, daß dieser genetische Algorithmus bezüglich *einer* Einstellung der Parameter auch für unterschiedliche Anwendungen gute oder optimale Ergebnisse erzielt. Es wird zusätzlich erreicht, daß die vorgestellten Ergebnisse reproduzierbar sind. Die Beurteilung der Qualität der Lösungen erfolgt durch einen Vergleich mit der von einem IP-Solver (hier: OSL [IBM92]) berechneten optimalen Lösung. Dazu werden die Ergebnisse in bezug auf die Anzahl der benutzten Instanzen der jeweiligen Bausteintypen sowie der Berechnungszeiten bewertet. Insbesondere der Vergleich der unterschiedlichen Laufzeiten ist wichtig, da hier die Motivation in der Durchführung der Mikroarchitektur-Synthese mit genetischen Algorithmen mitbegründet ist.

Alle im folgenden durchgeführten Tests beziehen sich auf eine Zykluszeit von 250 ns, Verbindungs- und Kontrollverzögerungen werden mit 20 ns berücksichtigt. Die im folgenden vorgestellten Testläufe sind auf einer SPARCstation20 (Taktfrequenz 60 MHz) durchgeführt worden. Für die Tests wird die folgende Bausteinbibliothek benutzt:

Bausteintyp	Ausführungszeit (in ns)	Kosten $k\lambda^2$
Addierer (Add1)	17	2405
Addierer (Add2)	31	1720
Multiplizierer (Mul1)	113	14717
Multiplizierer (Mul2)	251	11367
Subtrahierer (Sub)	17	2433

[1]Für GA1 erfolgte die Optimierung mit $seed = 1$ und für GA2 mit $seed = 2$ (siehe auch Anhang A).

Sowohl für die beiden Addierer Add1 und Add2 als auch für die beiden Multiplizierer Mul1 und Mul2 gilt, daß eine geringere physikalische Ausführungszeit gleichzeitig mit höheren Kosten (hier: Platzbedarf auf dem Chip) verbunden ist. Bis auf Mul2 ermöglichen alle Bausteintypen eine Ausführung der Operationen innerhalb eines Kontrollschrittes. Zur Durchführung einiger Tests sind Anpassungen der Bausteinbibliothek erforderlich, die an geeigneter Stelle beschrieben werden.

Im weiteren Verlauf dieses Kapitels werden die folgenden Untersuchungen für die angegebenen Benchmarks durchgeführt:

- Kontrollschritt-Erhöhung (EWF, FFT und FDCT)
- Chaining (EWF, FFT und FDCT)
- Berücksichtigung von Komplexbausteinen (EWF)
- Berücksichtigung von Pipeline-Bausteinen (EWF)
- Berücksichtigung von Zeitvorgabevorschriften und Verbindungsminimierung (EWF)

4.1 Untersuchungen bezüglich Kontrollschritt-Erhöhung

In diesem Abschnitt werden Testergebnisse für die drei Benchmarks EWF, FFT und FDCT bei einer Erhöhung der Anzahl zulässiger Kontrollschritte vorgestellt. In Tabelle 4.1 sind die Ergebnisse für den EWF (49 Operationen) von OSL, GA1 und GA2 dargestellt. Einträge in der Tabelle entsprechen der Anzahl benötigter Instanzen eines Bausteintyps.

EWF mit Kontrollschritt-Erhöhung																		
Anzahl cs	15			16			17			18			19			27		
Verfahren	OSL	GA1	GA2	OSL	GA1	GA2	OSL	GA1	GA2	OSL	GA1	GA2	OSL	GA1	GA2	OSL	GA1	GA2
Add1 (1 cs)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-	0	0
Add2 (1 cs)	3	3	3	3	3	3	2	2	3	2	2	3	2	2	2	-	2	2
Mul1 (1 cs)	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	-	1	1
Mul2 (2 cs)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-	0	0

Tabelle 4.1: Ergebnisse für den EWF bei Kontrollschritt-Erhöhung

Die von OSL berechneten Werte stellen optimale Ergebnisse dar (in der Tabelle heller gekennzeichnet). Es ist zu erkennen, daß bis auf zwei von GA2 berechnete Lösungen (siehe 17 und 18 Kontrollschritte) jeweils optimale Lösungen berechnet werden. Die Berechnung eines Ergebnisses von OSL bezüglich einer maximal erlaubten Anzahl von 27 Kontrollschritten konnte nicht innerhalb einer angemessenen Zeit erfolgen[2]. In Abbildung 4.1 werden die jeweiligen Ausführungszeiten vom IP-Solver OSL und der genetischen Algorithmen gegenübergestellt[3]. Aufgrund der

[2] Als eine angemessene Zeitspanne wird hier eine Rechenzeit von 100000 Sekunden (mehr als 27 Stunden) angenommen.

[3] Die Rechenzeiten der genetischen Algorithmen beziehen sich jeweils nur auf GA1, da nur unwesentliche Zeitunterschiede zu GA2 vorhanden sind.

großen Laufzeitunterschiede von OSL bezüglich der unterschiedlichen Beispiele wird des weiteren eine logarithmische Zeitskala verwendet.

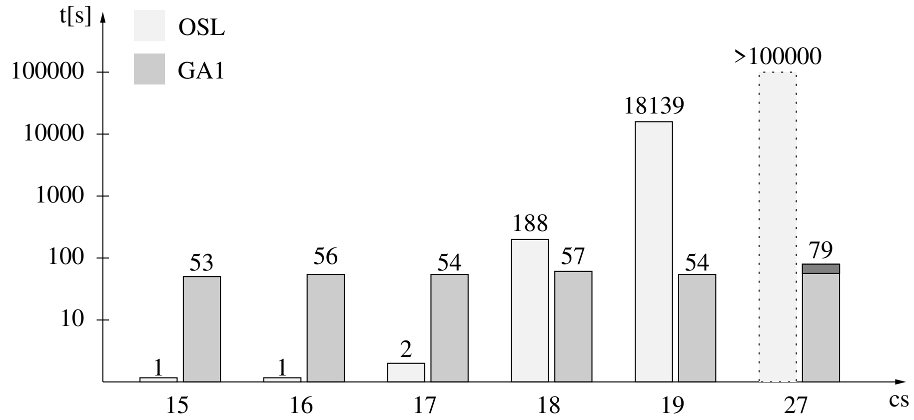


Abbildung 4.1: Rechenzeiten für den EWF bei Kontrollschritt-Erhöhung

Es zeigt sich, daß die Rechenzeit von OSL mit der Erhöhung von 17 auf 18 zulässige Kontrollschritte bereits deutlich ansteigt, während die von GA1 weitestgehend konstant bleibt. Lediglich bei der Berechnung der Lösung für 27 Kontrollschritte konnten merkliche Laufzeitunterschiede bei GA1 festgestellt werden, die aufgrund des Umfangs der IP-Datei und der damit verbundenen längeren Verifizierungsphase zustande kommt (im Diagramm durch einen dunkler markierten Bereich dargestellt). Da die Rechenzeiten von OSL bei einer Kontrollschritt-Erhöhung exponentiell steigen, wird für die restlichen Tests eine Beschränkung auf maximal zwei zusätzliche Kontrollschritte vorgenommen.

In Abbildung 4.2 werden die Testergebnisse mit den entsprechenden Rechenzeiten für die FFT (73 Operationen) vorgestellt.

FFT mit Kontrollschritt-Erhöhung									
Anzahl cs	9			10			11		
Verfahren	OSL	GA1	GA2	OSL	GA1	GA2	OSL	GA1	GA2
Add1 (1 cs)	0	0	0	0	0	0	0	0	0
Add2 (1 cs)	3	3	3	2	3	3	2	3	2
Mul1 (1 cs)	12	12	12	6	6	6	4	4	4
Mul2 (2 cs)	0	0	0	0	0	0	0	0	0
Sub (1 cs)	4	4	4	2	3	3	2	2	2

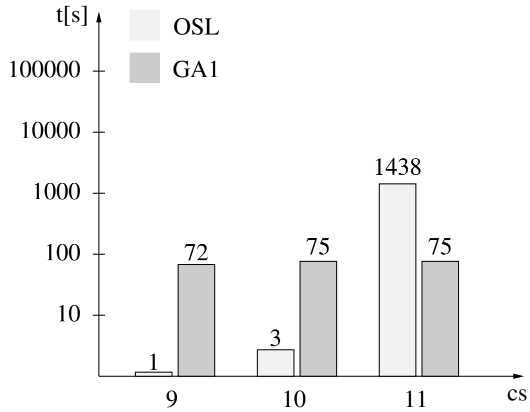


Abbildung 4.2: Ergebnisse und Rechenzeiten für die FFT bei Kontrollschritt-Erhöhung

Lediglich für eine Zeitschranke von 10 Kontrollschritten erzielen beide genetische Varianten keine optimale Lösung. Die Ausführungszeiten von OSL liegen bereits für 11 Kontrollschritte deutlich über denen von GA1.

Abbildung 4.3 gibt die Testergebnisse für die FDCT (66 Operationen) wieder.

FDCT mit Kontrollschritt-Erhöhung									
Anzahl cs	7			8			9		
Verfahren	OSL	GA1	GA2	OSL	GA1	GA2	OSL	GA1	GA2
Add1 (1 cs)	0	0	0	0	0	1	0	0	0
Add2 (1 cs)	4	4	4	4	4	3	2	3	3
Mul1 (1 cs)	8	8	8	5	4	5	4	4	4
Mul2 (2 cs)	0	0	0	0	2	0	0	0	0
Sub (1 cs)	2	3	2	2	2	2	2	2	2

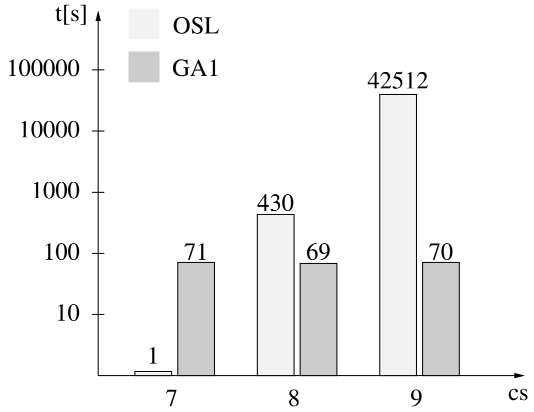


Abbildung 4.3: Ergebnisse und Rechenzeiten für die FDCT bei Kontrollschritt-Erhöhung

Für eine Zeitschranke von 8 und 9 Kontrollschritten werden keine optimalen Ergebnisse von GA1 und GA2 berechnet, stellen allerdings nahezu optimale Werte dar. Zum Beispiel besteht für GA2 bei der oberen Grenze von 8 Kontrollschritten der einzige Unterschied zur optimalen Lösung in der Auswahl des etwas teureren Addierers Add1 anstelle von Add2. Die im Diagramm dargestellten Ausführungszeiten bestätigen die bereits gewonnenen Erkenntnisse der vorherigen Beispiele. Für eine Zeitschranke von 8 Kontrollschritten liegen die Ausführungszeiten von OSL bereits deutlich über denen von GA1. In Abbildung 4.2 werden weitere Ergebnisse von GA1 und GA2 für die FDCT mit Kontrollschritt-Erhöhung dargestellt. Die Rechenzeiten bleiben für die hier vorgestellten Ergebnisse wiederum im wesentlichen konstant.

FDCT mit Kontrollschritt-Erhöhung																		
Anzahl cs	10		11		12		13		14		15		16		17		18	
Verfahren	GA1	GA2	GA1	GA2	GA1	GA2	GA1	GA2	GA1	GA2	GA1	GA2	GA1	GA2	GA1	GA2	GA1	GA2
Add1 (1 cs)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Add2 (1 cs)	3	3	2	2	2	2	2	3	2	2	2	2	2	2	2	2	2	2
Mul1 (1 cs)	3	3	3	3	3	2	2	1	2	2	2	2	2	1	2	2	2	2
Mul2 (2 cs)	0	0	0	0	0	1	0	2	0	0	0	0	0	1	0	0	0	0
Sub (1 cs)	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	2	2	1

Tabelle 4.2: Weitere Ergebnisse für die FDCT bei Kontrollschritt-Erhöhung

4.2 Berücksichtigung von Chaining

Die Berücksichtigung von Chaining führt zu einer weiteren Komplexitätssteigerung des Problems, so daß für einige der hier vorgestellten Ergebnisse trotz der zuvor gemachten Beschränkung auf zwei zusätzliche Kontrollschritte, keine Vergleichsergebnisse von OSL innerhalb von 100000 Sekunden erzielt werden. In den folgenden Testergebnissen (siehe Abbildungen 4.4

bis 4.6) zeigt sich für die drei Benchmarks, daß wiederum sehr gute Ergebnisse von GA1 und GA2 erzielt werden. Die Zulassung zusätzlicher Kontrollschritte führt in jedem der Beispiele zu einer Kostenreduzierung.

EWF mit Chaining und Kontrollschritt-Erhöhung									
Anzahl cs	6			7			8		
Verfahren	OSL	GA1	GA2	OSL	GA1	GA2	OSL	GA1	GA2
Add1 (17 ns)	1	2	2	-	2	2	-	1	1
Add2 (31 ns)	5	4	4	-	3	3	-	4	3
Mul1 (113 ns)	2	2	2	-	2	2	-	2	2
Mul2 (251 ns)	0	0	0	-	0	0	-	0	0

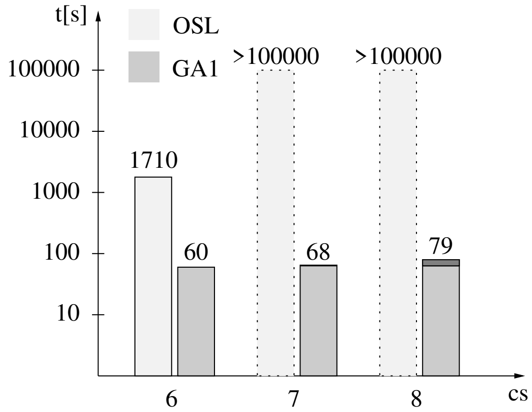


Abbildung 4.4: Ergebnisse und Rechenzeiten für den EWF bezüglich Chaining

FFT mit Chaining und Kontrollschritt-Erhöhung									
Anzahl cs	5			6			7		
Verfahren	OSL	GA1	GA2	OSL	GA1	GA2	OSL	GA1	GA2
Add1 (17 ns)	0	0	0	-	1	1	-	0	0
Add2 (31 ns)	5	5	5	-	4	4	-	3	3
Mul1 (113 ns)	12	12	12	-	6	6	-	4	4
Mul2 (251 ns)	0	0	0	-	0	0	-	0	0
Sub (17 ns)	5	5	5	-	4	4	-	3	3

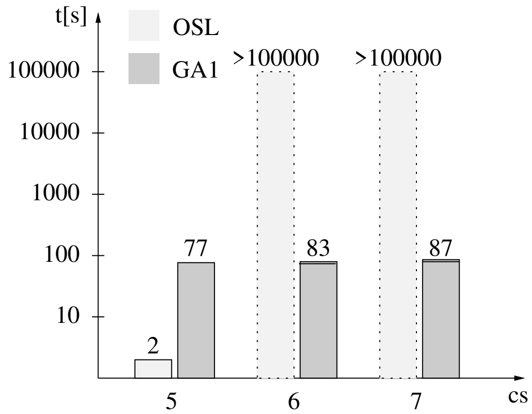


Abbildung 4.5: Ergebnisse und Rechenzeiten für die FFT bezüglich Chaining

FDCT mit Chaining und Kontrollschritt-Erhöhung									
Anzahl cs	4			5			6		
Verfahren	OSL	GA1	GA2	OSL	GA1	GA2	OSL	GA1	GA2
Add1 (17 ns)	1	1	1	-	1	0	-	1	0
Add2 (31 ns)	5	5	5	-	3	4	-	2	3
Mul1 (113 ns)	7	7	7	-	5	5	-	4	4
Mul2 (251 ns)	0	0	0	-	0	0	-	0	0
Sub (17 ns)	4	4	4	-	3	3	-	3	3

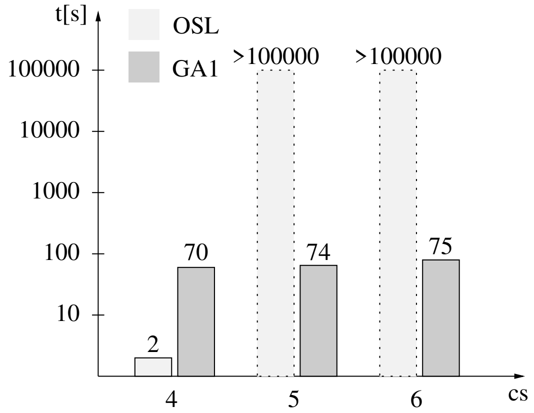


Abbildung 4.6: Ergebnisse und Rechenzeiten für die FDCT bezüglich Chaining

4.3 Verwendung von Komplexbausteinen

Untersuchungen mit Komplexbausteinen werden hier am Beispiel des EWF unter Verwendung eines MACs vorgestellt. Trotz höherer Kosten ($15312\, k\lambda^2$) gegenüber den anderen Bausteintypen erweist sich eine Verwendung dieses Bausteins als kostengünstigste Lösung, da auf dem MAC-Baustein sowohl Makrooperationen (Addition und Multiplikation) als auch die entsprechenden Einzeloperationen ausgeführt werden können (siehe Abbildung 4.7).

EWF mit MAC und Kontrollschritt-Erhöhung									
Anzahl cs	12			13			14		
Verfahren	OSL	GA1	GA2	OSL	GA1	GA2	OSL	GA1	GA2
Add1 (1 cs)	0	0	0	0	0	0	0	0	0
Add2 (1 cs)	2	2	2	2	1	2	2	2	1
Mul1 (1 cs)	0	0	0	0	0	0	0	0	0
Mul2 (2 cs)	0	0	0	0	0	0	0	0	0
MAC (1 cs)	2	2	2	1	2	2	1	1	2

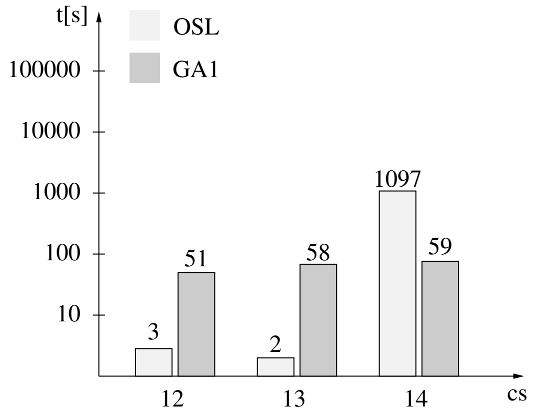


Abbildung 4.7: Ergebnisse und Rechenzeiten für den EWF mit MAC

Lediglich für eine Zeitschranke von 13 Kontrollschritten wird von GA1 und GA2 kein optimales Ergebnis berechnet.

4.4 Verwendung von Pipeline-Bausteinen

Nachfolgend werden Ergebnisse des EWF mit Pipeline-Bausteinen vorgestellt. Dazu wird die bisher benutzte Bausteinbibliothek dahingehend abgeändert, daß der einzyklische Multiplizierer (Mul1) durch einen Pipeline-Baustein (MulP) ersetzt wird. Der benutzte Pipeline-Baustein benötigt zur Ausführung einer Operation zwei Kontrollschritte und darf bereits nach einer Latenzzeit von einem Kontrollschritt mit einer neuen Operation beginnen. In Abbildung 4.8 wird dies durch die Notation 1:2 ausgedrückt.

EWF mit Pipeline-Baustein									
Anzahl cs	18			19			20		
Verfahren	OSL	GA1	GA2	OSL	GA1	GA2	OSL	GA1	GA2
Add1 (1 cs)	0	0	0	0	0	0	0	0	0
Add2 (1 cs)	3	3	3	3	3	3	2	2	3
MulP (1:2 cs)	1	1	1	1	1	1	1	1	1
Mul2 (2 cs)	1	1	1	0	1	1	0	0	0

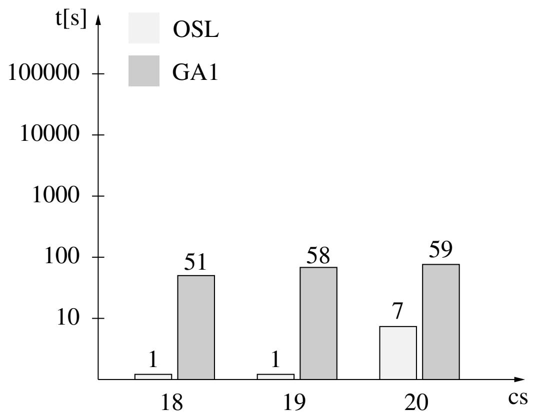


Abbildung 4.8: Ergebnisse und Rechenzeiten für den EWF bei Verwendung von Pipeline-Bausteinen

Die Testergebnisse zeigen, daß für eine erlaubte Anzahl von 18 und 20 Kontrollschritten optimale Ergebnisse von mindestens einem genetischen Algorithmus erzielt werden. Für die hier vorgestellte Testreihe ist im Vergleich zu den bisherigen ein geringerer Anstieg der Rechenzeiten von OSL zu erkennen.

4.5 Berücksichtigung von Zeitvorgabevorschriften und Verbindungsminimierung

Zunächst werden Tests unter Berücksichtigung von Zeitvorgabevorschriften vorgestellt. Dazu werden insgesamt fünf unterschiedliche Vorgaben zwischen Operationen in Form von minimalen, maximalen und konstanten Zeitabständen betrachtet. Abbildung 4.9 stellt das Ergebnis für den EWF dar.

EWF mit Zeitvorgabevorschriften			
Anzahl cs	15		
Verfahren	OSL	GA1	GA2
Add1 (1 cs)	0	0	0
Add2 (1 cs)	3	3	3
Mul1 (1 cs)	2	2	2
Mul2 (2 cs)	0	0	0

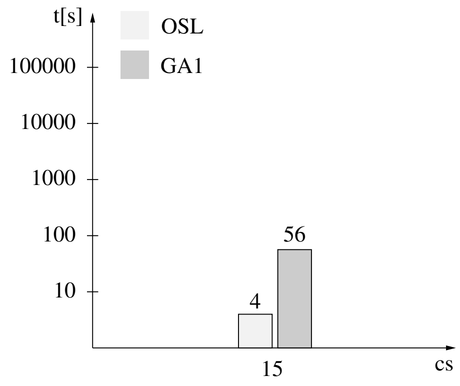


Abbildung 4.9: Ergebnis und Rechenzeit für den EWF mit Zeitvorgabevorschriften

Sowohl GA1 als auch GA2 berechnen das optimale Ergebnis.

Die Erzeugung von Ergebnissen unter Berücksichtigung von Verbindungsminimierung gestaltet sich aufgrund der extrem langen Rechenzeit von OSL als äußerst langwierig. Der Versuch, ein Ergebnis für eine Bausteinbibliothek mit 2 Addierern und 2 Multiplizierern zu berechnen, führt auch bei einer Rechenzeit von bis zu 50 Stunden zu keinem Ergebnis. Ebenso kann innerhalb derselben Zeitspanne mit einer reduzierten Bausteinbibliothek (ein einzyklischer Addierer und ein zweizyklischer Multiplizierer) kein Ergebnis berechnet werden. Erst mit einer weiteren Vereinfachung des Testproblems, indem statt des zweizyklischen Multiplizierers ein einzyklischer verwendet wird, kann ein Ergebnis nach mehr als 20 Stunden Rechenzeit von OSL berechnet werden. Das Ergebnis der Verbindungsminimierung ist in Tabelle 4.10 angegeben.

EWF mit Verbindungsminimierung			
Anzahl cs	15		
Verfahren	OSL	GA1	GA2
Anzahl Verbindungen	25	29	28
Add1 (1 cs)	3	3	3
Mul1 (1 cs)	2	2	2

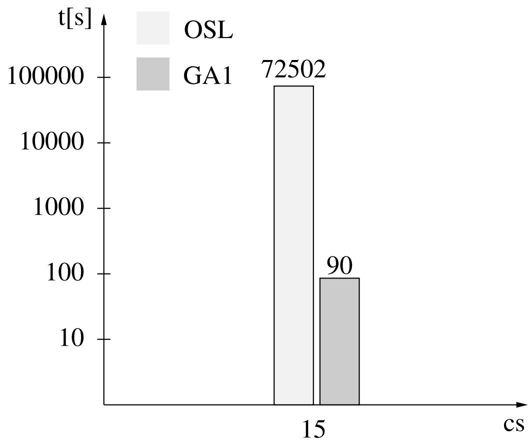


Abbildung 4.10: Ergebnisse und Rechenzeiten für den EWF mit Verbindungsminimierung

Die von GA1 und GA2 berechneten Ergebnisse stellen bis auf 3 beziehungsweise 4 zusätzlich benötigte Verbindungen optimale Ergebnisse dar. Im Vergleich zu den dargestellten Ergebnissen werden ohne Durchführung von Verbindungsminimierung 36 Verbindungen benötigt.

Die Darstellung der Testergebnisse in diesem Kapitel hat gezeigt, daß auch für unterschiedliche Beispiele viele optimale Ergebnisse von dem genetischen Algorithmus berechnet werden. Die Ausführungszeiten von GA1 und GA2 blieben dabei im wesentlichen konstant.

Kapitel 5

Zusammenfassung und Ausblick

5.1 Zusammenfassung

Im Rahmen dieser Diplomarbeit wurde ein genetischer Algorithmus zur Durchführung einer zeitbeschränkten Mikroarchitektur-Synthese entwickelt und implementiert. Aufgrund der Komplexität des Synthese-Problems wird auf die Forderung nach Optimalität der Ergebnisse zugunsten einer polynomiellen Laufzeit verzichtet. Diese Einschränkung ist erforderlich, damit auch für komplexere Problemstellungen möglichst optimale Lösungen in einer akzeptablen Zeit berechnet werden können. So ermöglichen Synthese-Verfahren, die auf der Basis der ganzzahlig linearen Programmierung beruhen, zwar die Berechnung global optimaler Ergebnisse bezüglich einer gegebenen Zielfunktion, führen allerdings bei der Lösung des aufgestellten Gleichungssystems zu exponentiellen Rechenzeiten des IP-Solver. Im Gegensatz dazu ist die Rechenzeit des hier vorgestellten Verfahrens im wesentlichen abhängig von der Anzahl der Operationen, so daß die Ausführungszeiten bei Kontrollschritt-Erhöhung nahezu konstant bleibt. Die in Kapitel 4 vorgestellten Testergebnisse der Benchmarks EWF (49 Operationen), FDCT (66 Operationen) und FFT (73 Operationen) bestätigen dies. Es konnte ebenfalls gezeigt werden, daß häufig optimale Ergebnisse berechnet werden.
Die Berücksichtigung der Verbindungsminimierung führt aufgrund der zusätzlich durchzuführenden Ressourcen-Bindung zu einer geringen Erhöhung der Rechenzeit. Dies ist jedoch in bezug auf die erzielte globale Betrachtung der drei Hauptaufgaben der Mikroarchitektur-Synthese mehr als vertretbar.

Mit der Anbindung an das OSCAR-System besteht die Möglichkeit, wahlweise die Vorteile eines IP-Solver oder die des genetischen Algorithmus zu nutzen. Als Schnittstelle dient hier in beiden Fällen eine vom OSCAR-System generierte IP-Datei. Anhand dieser Datei ist ebenfalls eine Verifizierung der berechneten Lösung im Sinne der Correctness-by-Construction möglich.

Die simultan zur Minimierung der Hardware-Kosten durchgeführte Reduzierung der Anzahl benötigter Kontrollschritte einer Schaltung führt in der Regel nur zu geringen Zeiteinsparungen[1]. Hierzu durchgeführte Untersuchungen von Heijligers ergaben für die in [DHSB95, WGH90] vorgestellten genetischen Algorithmen seiner Meinung nach „enttäuschende Resultate“ (siehe [Hei96] Seite 124). Eine effektive Möglichkeit zur Ermittlung der kürzesten Ausführungszeit für einen bestimmten Hardware-Aufwand besteht allerdings in einer schrittweisen Erhöhung der Anzahl erlaubter Kontrollschritte.

[1]Für das in Tabelle 4.1 angegebene Testergebnis mit 27 Kontrollschritten wird zum Beispiel von GA1 eine Lösung mit 25 und von GA2 mit 24 Kontrollschritten Ausführungszeit berechnet. Optimal sind bei diesem Hardware-Aufwand 17 Kontrollschritte.

Mit den bisher existierenden Verfahren auf der Basis genetischer Algorithmen konnte bislang nur ein sehr geringer Funktionsumfang realisiert werden. Der in dem hier vorgestellten genetischen Algorithmus realisierte umfangreiche Funktionsumfang stellt einen wesentlichen Vorteil gegenüber diesen Verfahren dar. Dies sind im einzelnen:

- Unterstützung umfangreicher Bausteinbibliotheken
  - Bausteine unterschiedlicher Ausführungsgeschwindigkeiten
  - Mehrzyklische Bausteine
  - Komplexbausteine
  - Pipeline-Bausteine
- Berücksichtigung von Chaining
- Unterstützung von minimalen, maximalen und konstanten Zeitvorgabevorschriften zwischen Operationen
- Durchführung von Verbindungsminimierung

Zusätzlich wird die Anzahl der benötigten Kontrollschritte einer Schaltung durch eine Erweiterung der Kostenfunktion berücksichtigt.

5.2 Ausblick

Der hier beschriebene genetische Algorithmus stellt bereits ein leistungsfähiges Verfahren zur Durchführung der Mikroarchitektur-Synthese dar. Dennoch bestehen Verbesserungs- und Erweiterungsmöglichkeiten, von denen einige nachfolgend angegeben werden:

- Erweiterungen des Funktionsumfangs:
  Zusätzliche IP-Nebenbedingungen können zum Beispiel in Form von (Un-) Gleichungen in die IP-Datei aufgenommen werden. Zur Bewertung eines Individuums besteht dann die Möglichkeit, diese (Un-) Gleichungen auf Gültigkeit zu testen. Dabei wird die Anzahl der Verstöße bei der Bewertung des jeweiligen Individuums mitberücksichtigt.
- Dynamische Kontrolle der Parameter des genetischen Algorithmus:
  Eine Anpassung der internen Parameter des genetischen Algorithmus an aktuelle Gegebenheiten, wie in [LT93] mittels eines Fuzzy-Experten vorgestellt, kann zu einer weiteren Verbesserung der Ergebnisse führen. Zum Beispiel ist es sinnvoll, im Falle einer vorzeitigen Homogenisierung der Population die Mutationsrate zu erhöhen, um neues Genmaterial in die Population einzubringen. Weiterhin ist es denkbar, daß durch eine entsprechende dynamische Anpassung der Gewichtung benötigter Kontrollschritte einer Schaltung Verbesserungen bei einer gleichzeitigen Minimierung von Kosten und Ausführungszeit erzielt werden.
- Durchführung von Tests bezüglich unterschiedlicher Mutations- und Crossover-Varianten:
  Zum Beispiel wird bei dem in [ASB94] vorgestellten genetischen Algorithmus eine Crossover-Variante vorgestellt, bei der nur Gene von Operationen ausgetauscht werden, die zu keiner Verletzung der Datenabhängigkeiten führen. So kann in [ASB94] aufgrund des geringen Funktionsumfangs ein korrektheitserhaltendes Crossover realisiert werden.

- Verwendung zusätzlicher Heuristiken:
  Bei der Allokation von Bausteintypen kann es vorkommen, daß bei gleicher Funktionalität nicht der billigste Bausteintyp ausgewählt wird (siehe Testergebnis in Abbildung 4.3 für GA2 bei 8 Kontrollschritten). Falls ausreichend Instanzen des billigeren Bausteintyps vorhanden sind, kann dies von vornherein verhindert werden, indem keine Zuweisungen von Operationen zu teureren Bausteintypen vorgenommen werden. Eine weitere Möglichkeit besteht in einer zusätzlichen Nachverarbeitungsphase, in der entsprechende Operationen zu einer freien Instanz des billigeren Bausteintyps zugewiesen werden.
  Bei der Einbindung zusätzlicher Heuristiken muß allerdings darauf geachtet werden, daß das Verhalten des genetischen Algorithmus nicht zu deterministisch wird.

- Parallelisierung von Berechnungen:
  In genetischen Algorithmen lassen sich viele Vorgänge parallelisieren, da für die Individuen viele Berechnungen unabhängig voneinander durchgeführt werden können. Durch eine Verteilung der Berechnungen auf mehrere Prozessoren kann hierdurch eine drastische Verkürzung der Rechenzeit erreicht werden.

Anhang A

Benutzerhandbuch

An dieser Stelle wird beschrieben, in welcher Weise der Designer Einfluß auf den Ablauf des genetischen Algorithmus nehmen kann. Dies kann zum Beispiel sinnvoll sein, falls für ein Design mehrere Optimierungsläufe durchgeführt werden sollen, um von diesen das beste Ergebnis auszuwählen. So besteht die Möglichkeit, einerseits die standardmäßig vorgegebene Zufallszahlenfolge zu ändern und andererseits die Paramater des genetischen Algorithmus problemspezifisch anzupassen.

Für eine vom OSCAR-System generierte IP-Datei <ip_datei> lautet der allgemeine Aufruf des entwickelten Gesamtsystems folgendermaßen:

`gen_syn` [optionen] `<` <ip_datei>

Änderungen der vorgegebenen Standardeinstellungen der Parameter können durch die folgenden Erweiterungen des Funktionsaufrufs erfolgen:

-p_cross <parameter>: Crossover-Wahrscheinlichkeit $prob_{cross} \in [0, 1]$ zweier Eltern.
Standard: 0.4

-p_uni <parameter>: Austauschwahrscheinlichkeit $prob_{uni} \in [0, 1]$ eines Gens beim Uniform Crossover.
Standard: 0.7

-p_m <parameter>: Wechselwahrscheinlichkeit $prob_m \in [0, 1]$ eines Bausteintyps bei Durchführung der Allokation.
Standard: 0.7

-p_cs <parameter>: Mutationswahrscheinlichkeit $prob_{cs} \in [0, 1]$ für die Belegung des Kontrollschritt-Gens einer Operation.
Standard: $2/j_{max}$

-p_dir <parameter>: Wahrscheinlichkeit $prob_{dir} \in [0, 1]$, mit der eine Operation im Falle einer Mutation einem früheren Ausführungszeitpunkt zugewiesen wird.
Standard: 0.6

-p_hw <parameter>: Mutationswahrscheinlichkeit $prob_{hw} \in [0, 1]$ für die Belegung des Hardware-Gens einer Operation.
Standard: $2/j_{max}$

-p_mac <parameter>: Wahrscheinlichkeit $prob_{mac} \in [0, 1]$, mit der ein Wechsel zwischen der Realisierung einer Makrooperation und den dazugehörigen Einzeloperationen durchgeführt wird.
Standard: 0.009

-w_cs <parameter>: Gewichtung $w_{cs} \in \mathbb{R}$, mit der jeder zur Ausführung der Schaltung erforderliche Kontrollschritt in die Gesamtkosten eingeht.
Standard: $0.00001 * (cost_{max}/i_{max})$

-num_gen <parameter>: Anzahl der Generationen $num_{gen} \in \mathbb{N}$ nach der die Optimierung des genetischen Algorithmus terminiert.
Standard: 50

-popsize <parameter>: Anzahl der Individuen $popsize \in \mathbb{N}$ in einer Generation.
Standard: 100

-num_repl <parameter>: Anzahl der Individuen $num_{repl} \in \{1, \ldots, popsize\}$, die ohne Veränderung in die nächste Generation übernommen werden.
Standard: 50

-seed <parameter>: Parameter $seed \in \mathbb{N}$ zur Bestimmung einer Zufallszahlenfolge.
Standard: 1

Beispiel:
Für eine Populationsgröße von 60 Individuen und eine Crossover-Wahrscheinlichkeit von 0.6, bei unveränderten restlichen Parametern, lautet der Aufruf des Gesamtsystems:

```
gen_syn -popsize 60 -p_cross 0.6 < <ip_datei>
```

Anhang B

Notationen

Notation	Erklärung
$I \subset \mathbb{N}\ ,\ i \in I$	Menge der Kontrollschritte $i \in I = \{1, \ldots, i_{max}\}$
$J \subset \mathbb{N}\ ,\ j \in J$	Menge der Operationen $j \in J = \{1, \ldots, j_{max}\}$
$K \subset \mathbb{N}\ ,\ k \in K$	Menge der Instanzen $k \in K = \{1, \ldots, k_{max}\}$
$M \subset \mathbb{N}\ ,\ m \in M$	Menge der Bausteintypen $m \in M = \{1, \ldots, m_{max}\}$
$\mathrm{ASAP}(j) \in I$	frühestmöglicher Kontrollschritt, zu dem mit der Ausführung von Operation j begonnen werden darf
$\mathrm{ALAP}(j) \in I$	spätestmöglicher Kontrollschritt, zu dem mit der Ausführung von Operation j begonnen werden muß
$\mathrm{ASAP}_{dyn}(j) \in I$	frühestmöglicher dynamischer Kontrollschritt, zu dem mit der Ausführung von Operation j begonnen werden darf $\mathrm{ASAP}_{dyn}(j) \geq \mathrm{ASAP}(j)$
$\mathrm{ALAP}_{dyn}(j) \in I$	spätestmöglicher dynamischer Kontrollschritt, zu dem mit der Ausführung von Operation j begonnen werden muß $\mathrm{ALAP}_{dyn}(j) \leq \mathrm{ALAP}(j)$
$CS(j) \in I$	Kontrollschritt, zu dem die Ausführung von Operation j gestartet wird
$HW(j) \in \mathbb{N}_0$	Hardware-Baustein (Instanz oder Register), dem Operation j zugewiesen wird

Tabelle B.1: Allgemeine Notationen (Teil a)

Notation	Erklärung
$C(j,k) \in \mathbb{N}_0$	Ausführungszeit (in Kontrollschritten), die Instanz k zur Ausführung von Operation j benötigt
$\ell(j,k) \in \mathbb{N}_0$	Latenzzeit (in Kontrollschritten), die Instanz k nicht mit einer Operation belegt werden darf, nachdem die Ausführung von Operation j auf Instanz k gestartet wurde
$t_{chain}(j,k) \in \mathbb{R}^+$	in den Chaining-Nebenbedingungen angegebene Ausführungszeit (in ns) von Operation j auf Instanz k zuzüglich einer Verbindungsverzögerung
$ch_{accu}(j) \in \mathbb{R}^+$	kumulierte Ausführungszeit der zeitmäßig längsten Operationen-Kette in Kontrollschritt $\mathrm{ASAP}_{dyn}(j)$, mit der Operation j eine Chaining-Verbindung eingehen kann
$t_{remain}(j,i) \in \mathbb{R}^+$	verbliebene Ausführungszeit (in ns), falls Operation j in Kontrollschritt i ausgeführt wird
$t_{cs} \in \mathbb{R}^+$	in den Chaining-Nebenbedingungen angegebene Zykluszeit (in ns) abzüglich einer Kontrollverzögerung
$j \prec j'$	Datenabhängigkeit der Operation j' von Operation j
$j \prec\!\!\prec j'$	Chaining-Möglichkeit zwischen Operation j und Operation j'
$min_{timing}(j,j') \in \mathbb{N}_0$	Mindestabstand (in Kontrollschritten), der zwischen den Operationen j und j' eingehalten werden muß
$max_{timing}(j,j') \in \mathbb{N}_0$	Maximalabstand (in Kontrollschritten), der zwischen den Operationen j und j' auftreten darf
$type(k) \in M$	Bausteintyp von Instanz k
$macro_{flag}(j) \in \mathbb{N}_0$	Markierung der Operation j zur Realisierung des gegenseitigen Ausschlusses einer Makrooperation und den dazugehörigen n Einzeloperationen
$c_k \in \mathbb{R}^+$	Kosten, die Instanz k verursacht
$c_{k,k'} \in \mathbb{R}^+$	Kosten, die für eine Verbindung von Instanz k zu Instanz k' entstehen
$cost_{max} \in \mathbb{R}$	Kostenobergrenze für einen Entwurf
$w_{pen} \in \mathbb{R}$	Gewicht, mit dem das Ausmaß des Fehlers $penalty(ind)$ von Individuum ind in die Kostenfunktion eingeht
$ind \in \{1,\ldots,popsize\}$	Individuum einer Population
$penalty(ind) \in \mathbb{N}$	Ausmaß des Fehlers von Individuum ind
$num_{cs} \in \{1,\ldots,i_{max}\}$	Anzahl der benötigten Kontrollschritte eines Entwurfs
$b_k \in \{0,1\}$	binäre Entscheidungsvariable: 1, falls die Instanz k benutzt wird 0, sonst
$w_{k,k'} \in \{0,1\}$	binäre Entscheidungsvariable: 1, falls eine Verbindung zwischen Instanz k und Instanz k' existiert 0, sonst

Tabelle B.2: Allgemeine Notationen (Teil b)

Notation	Erklärung
$tab_{bind}(k,i)$	Tabelle zur Verwaltung aller Instanzen Dimension: (k_{max}, i_{max})
$tab_{alloc}(m,i)$	Tabelle zur Ermittlung der Anzahl benötigter Instanzen eines Bausteintyps m in Kontrollschritt i Dimension: (m_{max}, i_{max})
$tab_{inter}(k,k)$	Tabelle zur Berechnung der benötigten Verbindungen zwischen den einzelnen Instanzen Dimension: (k_{max}, k_{max})
$cs_{array}(j)$	Kontrollschritt-Array für Operation j, um gezielt Kontrollschritte innerhalb von $\{\text{ASAP}(j), \ldots, \text{ALAP}(j)\}$ ungültig setzen zu können Dimension: $(1, \text{ALAP}(j))$
$succ_{list}(j)$	Liste mit allen direkten Nachfolgeoperationen von Operation j
$timing_{list}(j)$	Liste mit Zeitvorgabevorschriften, die von Operation j zu Operationen mit einer höheren Ordnungsnummer bestehen

Tabelle B.3: Verwendete Datenstrukturen

Notation	Erklärung
$prob_{cross} \in [0,1]$	Wahrscheinlichkeit, mit der zwischen zwei selektierten Eltern ein Crossover stattfindet
$prob_{uni} \in [0,1]$	Wahrscheinlichkeit, mit der ein Gen zwischen zwei Eltern ausgetauscht wird
$prob_{m} \in [0,1]$	Wahrscheinlichkeit, mit der ein Bausteintyp für eine Operation bei der Allokation gewechselt wird
$prob_{cs} \in [0,1]$	Wahrscheinlichkeit, mit der die Belegung des Kontrollschritt-Gens einer Operation geändert wird
$prob_{dir} \in I\!R$	Wahrscheinlichkeit, mit der eine Operation einem früheren Ausführungszeitpunkt zugewiesen wird, falls eine Mutation des Kontrollschrittes erfolgen soll
$prob_{hw} \in [0,1]$	Wahrscheinlichkeit, mit der die Belegung des Hardware-Gens einer Operation geändert wird
$prob_{mac} \in [0,1]$	Wahrscheinlichkeit, mit der ein Wechsel zwischen der Realisierung einer Makrooperation und den dazugehörigen Einzeloperationen durchgeführt wird
$w_{cs} \in I\!R$	Gewicht, mit dem die Anzahl der von einem Entwurf benötigten Kontrollschritte (num_{cs}) in die Kostenfunktion eingeht
$num_{gen} \in I\!N$	Anzahl der Generationen, die im genetischen Algorithmus durchlaufen werden, bevor das beste erzeugte Individuum ausgegeben wird
$popsize \in I\!N$	Anzahl der Individuen in einer Generation
$num_{repl} \in \{1, \ldots, popsize\}$	Anzahl der Individuen, die ohne Veränderung in die nächste Generation übernommen werden
$seed \in I\!N$	Parameter zur Beeinflussung der Zufallszahlenfolge des genetischen Algorithmus

Tabelle B.4: Vom Designer spezifizierbare Größen

Literaturverzeichnis

[Ach95] H. Achatz. Datenpfadsynthese mit Hilfe von ganzzahliger linearer Optimierung. *Dissertation, Universität Passau*, 1995.

[ASB94] S. Ali, S. M. Sait, and M. S. T. Benten. GSA: Scheduling and Allocation using Genetic Algorithm. *Proceedings of the EURO-DAC*, pages 84–89, 1994.

[Bak74] K. R. Baker. *Introduction to Sequencing and Scheduling.* Wiley, New York, 1974.

[CT90] R. J. Cloutier and D. E. Thomas. The Combination of Scheduling, Allocation, and Mapping in a Single Algorithm. *Proceedings of the 27th Design Automation Conference*, pages 71–76, 1990.

[DHSB95] M. K. Dhodhi, F. H. Hielscher, R. H. Storer, and J. Bhasker. Datapath Synthesis Using a Problem-Space Genetic Algorithm. *IEEE Transactions on CAD, Vol. 14, No. 8*, pages 934–943, 1995.

[Döm94] R. Dömer. Optimale Mikroarchitektursynthese mittels Ganzzahliger Programmierung. *Diplomarbeit, Universität Dortmund*, 1994.

[GE91] C. H. Gebotys and M. I. Elmasry. Simultaneous Scheduling and Allocation for Cost Constraint Optimal Architectural Synthesis. *Proceedings of the 28th Design Automation Conference*, pages 2–7, 1991.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* Freeman and Company, New York, 1979.

[GK83] D. D. Gajski and R. H. Kuhn. New VLSI Tools. *IEEE Computer, Vol. 16, No. 12*, pages 11–14, 1983.

[Gol89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison Wesley, 1989.

[Hei94] J. Heistermann. *Genetische Algorithmen.* Teubner Verlag, 1994.

[Hei96] M. J. M. Heijligers. The Application of Genetic Algorithms to High-Level Synthesis. *Ph. D. Thesis, Technische Universiteit Eindhoven*, 1996.

[Hol92] J. H. Holland. *Adaption in Natural and Artificial Systems.* MIT Press, 1992.

[IBM92] IBM Corp. Optimization Subroutine Library (osl), Guide and Reference, 1992.

[IEE88] Design Automation Standards Subcommittee of the IEEE. IEEE Standard VHDL Language Reference Manual (IEEE Std. 1076-87). *IEEE Inc., New York*, 1988.

[KKB91] A. Kumar, A. Kumar, and M. Balakrishnan. A novel integrated scheduling and allocation algorithm for data path synthesis. *Proceedings of VLSI-Design, New Delhi*, pages 212–218, 1991.

[KP87] F. J. Kurdahi and A. C. Parker. REAL: A Program for REgister ALlocation. *Proceedings of the 24th ACM/IEEE Design Automation Conference*, pages 210–215, 1987.

[KWK85] S. Y. Kung, H. J. Whitehouse, and T. Kailath. *VLSI and Modern Signal Processing.* Prentice Hall, 1985.

[Lan97] B. Landwehr. ILP-basierte Mikroarchitektur-Synthese mit komplexen Bausteinbibliotheken. *Dissertation, Universität Dortmund*, 1997.

[LM93] T. A. Ly and J. T. Mowchenko. Applying Simulated Evolution to High Level Synthesis. *IEEE Transactions on CAD, Vol. 12, No. 3*, pages 389–409, 1993.

[LMD94] B. Landwehr, P. Marwedel, and R. Dömer. OSCAR: Optimum Simultaneous Scheduling, Allocation and Resource Binding Based on Integer Programming. *Proceedings of the EURO-DAC*, pages 90–95, 1994.

[LMMD97] B. Landwehr, P. Marwedel, I. Markhof, and R. Dömer. Exploiting Isomorphism for Speeding-Up Instance-Binding in an Integrated Scheduling, Allocation and Assignment Approach to Architectural Synthesis. *Conference on Computer Hardware Description Languages and their Applications*, 1997.

[LT93] M. A. Lee and H. Takagi. Dynamic Control of Genetic Algorithms using Fuzzy Logic Techniques. *Proceedings of Fifth Int. Conf. on Genetic Algorithms, Urbana-Champaign*, pages 76–83, 1993.

[Man95] C. A. Mandal. Complexity Analysis and Algorithms for Data Path Synthesis. *Ph. D. Thesis, Dpt. of Computer Science and Engineering, Indian Institute of Technology, Kharagpur*, 1995.

[Mar93] P. Marwedel. *Synthese und Simulation von VLSI-Systemen.* Carl Hanser Verlag, 1993.

[MD90] D. J. Mallon and P. B. Dexyer. A New Approach To Pipeline Optimization. *IFIP Working Conference on Logic and Architecture*, pages 83–88, 1990.

[MLD96] P. Marwedel, B. Landwehr, and R. Dömer. Built-in Chaining: Introducing Complex Components into Architectural Synthesis. Forschungsbericht 611, Universität Dortmund, 1996.

[MLD97] P. Marwedel, B. Landwehr, and R. Dömer. Built-in Chaining: Introducing Complex Components into Architectural Synthesis. *Proceedings of the ASP-DAC*, 1997.

[Nis97] V. Nissen. *Einführung in Evolutionäre Algorithmen.* Vieweg, Wiesbaden, 1997.

[PK87] P. G. Paulin and J. P. Knight. Force-Directed Scheduling in Automatic Data Path Synthesis. *Proceedings of the 24th Design Automation Conference*, pages 195–202, 1987.

[PS82] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization. Algorithms and Complexity.* Prentice Hall, 1982.

[Sei95] H. Seidel. XOSCAR - Eine graphische Oberfläche zur interaktiven Unterstützung der Mikroarchitektursynthese. *Diplomarbeit, Universität Dortmund*, 1995.

[WGH90] N. Wehn, M. Glesner, and M. Held. A Novel Scheduling/Allocation Approach for Datapath Synthesis based on Genetic Paradigms. *IFIP Workshop on Logic and Architecture Synthesis, Paris*, pages 47–56, 1990.