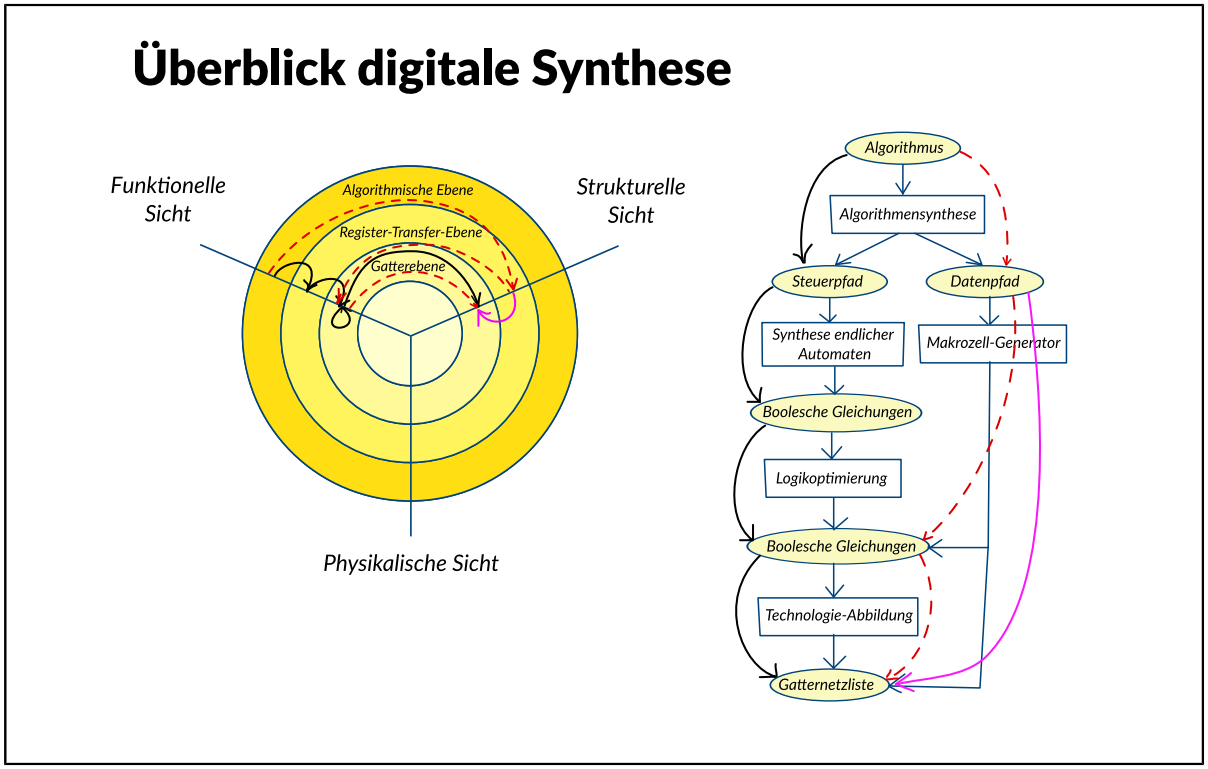
Electronic Design Automation (EDA)

Algorithmensynthese

Date of generation: 30.5.2023

Algorithmensynthese: Überblick digitale Synthese



Aufgabe der digitalen Synthese ist die Übersetzung einer funktionellen Schaltungsbeschreibung in einer Hardware-Beschreibungssprache (HDL, z. B. Verilog, VHDL) in eine Gatternetzliste. Ausgangspunkt der digitalen Synthese im Y-Diagramm ist die algorithmische Ebene in der funktionellen Sicht, am Ende liegt die Schaltung auf der Gatterebene in struktureller Sicht vor.

Die digitale Synthese gliedert sich in vier Schritte, die in der Abbildung dargestellt sind:

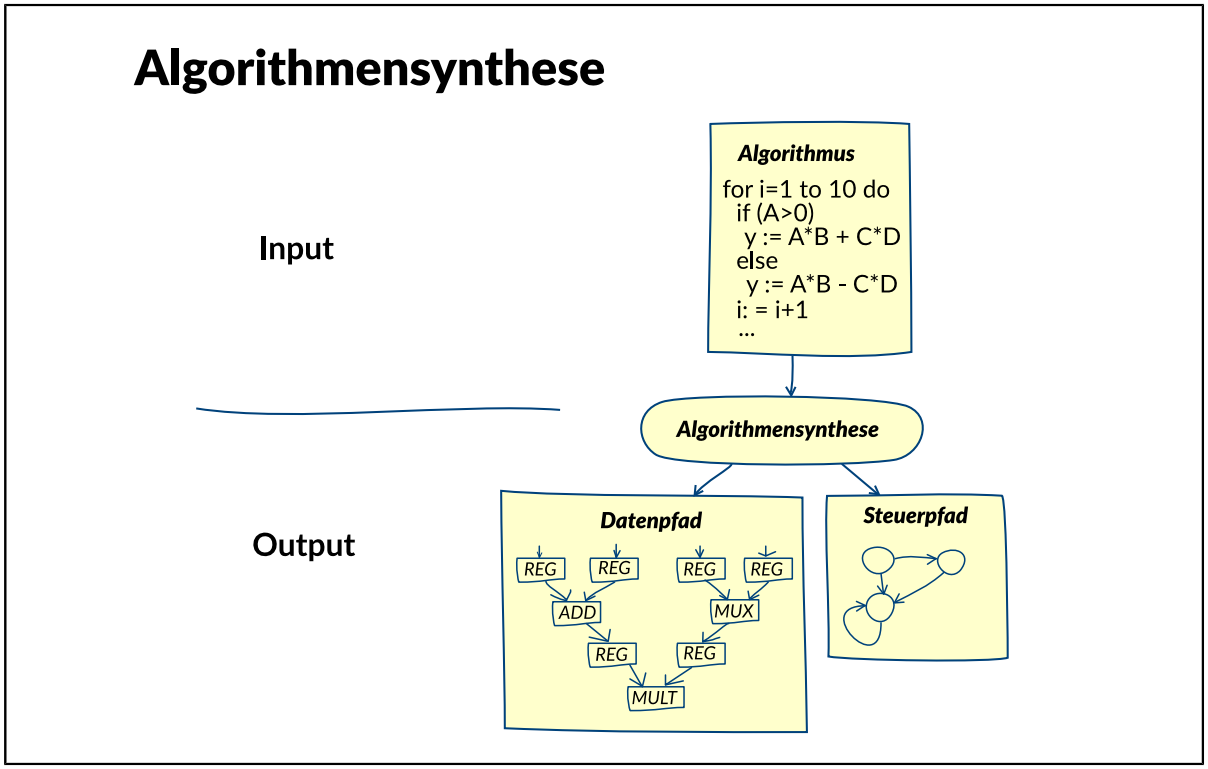
- **Algorithmensynthese**
  Hier wird aus der Verhaltensbeschreibung in einer HDL eine Register-Transfer-Struktur erzeugt. Dabei wird üblicherweise die Schaltung in einen Datenpfad und einen Steuerpfad zerlegt. Der Datenpfad beschreibt die vom Algorithmus auszuführenden arithmetischen Operationen, während der Steuerpfad den Ablauf des Algorithmus kontrolliert und als Zustandsübergangsdiagramm beschrieben werden kann. Für den Datenpfad findet bereits in diesem Schritt eine Optimierung von Kosten bzw. Performance statt.
- **Register-Transfer-Synthese**
  Hier werden der Daten- und der Steuerpfad getrennt voneinander synthetisiert. Der Steuerpfad, der als endlicher Automat (Finite State Machine, FSM) dargestellt werden kann, wird durch boolesche Gleichungen (Gatter) und Speicherelemente realisiert. Der Datenpfad wird ebenfalls mit optimierten booleschen Gleichungen (Gatter) und Speicherelementen (Register) realisiert. Für stark spezialisierte Strukturen wie Speicherfelder oder arithmetische Elemente wie Addierer oder Multiplizierer können durch Makrozellgeneratoren technologieabhängige Beschreibungen wie eine Gatternetzliste oder ein Layout erzeugt werden.
- **Logikoptimierung**
  Die im Rahmen der bisher ausgeführten Schritte entstandenen booleschen Gleichungen weisen noch ein erhebliches Optimierungspotential auf. Entsprechend der Kriterien Performance und Kosten können die logische Schaltungstiefe und die Anzahl der benötigten Gatter optimiert werden.
- **Technologie-Abbildung auf Logikebene**
  Das nach der Optimierung erhaltene System boolescher Gleichungen ist noch unabhängig von der Technologie, in der die Schaltung realisiert werden soll. In diesem Schritt wird es mit Hilfe einer

technologieabhängigen Bibliothek von Grundgattern in eine Gatternetzliste umgesetzt. Die Gatter sind hinsichtlich Fläche und Zeitverhalten charakterisiert.

Diese Schritte enthalten erneut Optimierungen, bei denen Kompromisse zwischen Performance und Kosten eingegangen werden. Ein weiteres Optimierungsziel kann der Leistungsverbrauch der Schaltung sein.

Aufgrund des hohen Abstraktionsgrads der Schaltungsbeschreibung auf algorithmischer und Register-Transfer-Ebene sind die Optimierungskriterien nur durch stark vereinfachende Modelle beschreibbar. Die Genauigkeit der Modelle nimmt auf den unteren Abstraktionsebenen zu. Daher kann oft erst nach der Technologie-Abbildung festgestellt werden, ob bestimmte Ziele von vorausgehenden Synthese-Schritten erreicht wurden. Dann ist ein erneutes Durchführen dieser Schritte erforderlich.

Algorithmensynthese: Algorithmensynthese



Die Algorithmensynthese verwendet als Eingabedaten eine generische Beschreibung des zu synthetisierenden Algorithmus. Für die Beschreibung wird eine sequenzielle Abfolge von Operationen in einer Hochsprache verwendet. Hardwarebeschreibungssprachen bieten die Möglichkeit, das sequenzielle Ablaufschema eines Algorithmus zu beschreiben.

Das Ziel der Algorithmensynthese ist die Umsetzung der gegebenen Verhaltensbeschreibung in eine äquivalente strukturelle Beschreibung auf RT-Ebene. Die zu erreichende Struktur besteht dabei aus einem so genannten Datenpfad und einem zugehörigen Steuerpfad. Der Datenpfad enthält alle vom Algorithmus benötigten arithmetischen Operatoren sowie Multiplexer und Register zur Datenzuführung und Zwischenspeicherung. Die Operatoren werden als arithmetische Makrozellen instanziiert. Mit Hilfe des Steuerpfads erfolgt die Ansteuerung des Datenpfads, so dass die Operationen in der durch die Verhaltensbeschreibung vorgegebenen Reihenfolge ausgeführt werden. Der Steuerpfad wird als endlicher Automat realisiert, dessen Zustandsübergangsdiagramm an den nächsten Syntheseschritt zur weiteren Verarbeitung übergeben wird.

Algorithmensynthese: Beispiel für einen Algorithmus

Beispiel für einen Algorithmus

Ziel: schrittweise Lösung der Differentialgleichung

$$\frac{d^2u}{dt^2} + 2\,\frac{du}{dt}\,t + 10u = 0$$

DGL

VHDL

```
entity DGL is
port( t_in, u_in, s_in, dt_in, T_in : IN REAL;
      u_out : OUT REAL; start : in std_logic);
end DGL;

architecture behv of DGL is
BEGIN
dgl_1: process( start)
VARIABLE t, u, s, dx, T, t1, s1, u1 : REAL;
   BEGIN
   t:= t_in; u:=u_in; s:=s_in; dt:=dt_in; T:=T_in;
   LOOP
     t1 := t+dt;
     u1 := u+(s*dt);
     s1 := s -(2*t*(s*dt))-(10*u*dt);
     t:=t1;   u:=u1;   s:= s1;
    EXIT WHEN t1>T;
  END LOOP;
  u_out <= u;
  end dlg_1;
end behv;
```

Beim Betrieb eines Vorgängermodells der Ampelsteuerung hat sich herausgestellt, dass aufgrund parasitäter Kapazitäten und Induktivitäten in den Signalleitungen Schwingungen auftreten, die durch eine nichtlineare Differentialgleichung zweiter Ordnung beschrieben werden können:

$$\frac{d^2u}{dt^2} + 2\,\frac{du}{dt}\,t + 10u = 0.$$

Die Spannung u soll deshalb während des Betriebes vorausgesagt werden können. Dazu soll ein entsprechender Algorithmus, der die Differentialgleichung in einem Zeitintervall T schrittweise numerisch löst, aus Performance-Gründen in einem spezifischen IC hardwaremäßig implementiert werden. Der dazu gehörige Algorithmus ist im Bild dargestellt. Nach Eingabe der Anfangswerte für t, u, s (=du/dt), der Schrittweite dt sowie des Intervalls T berechnet dieser den Spannungswert u am Ende des Intervalls gemäß folgender Rechnung:

t = t + dt

s = (du / dt)

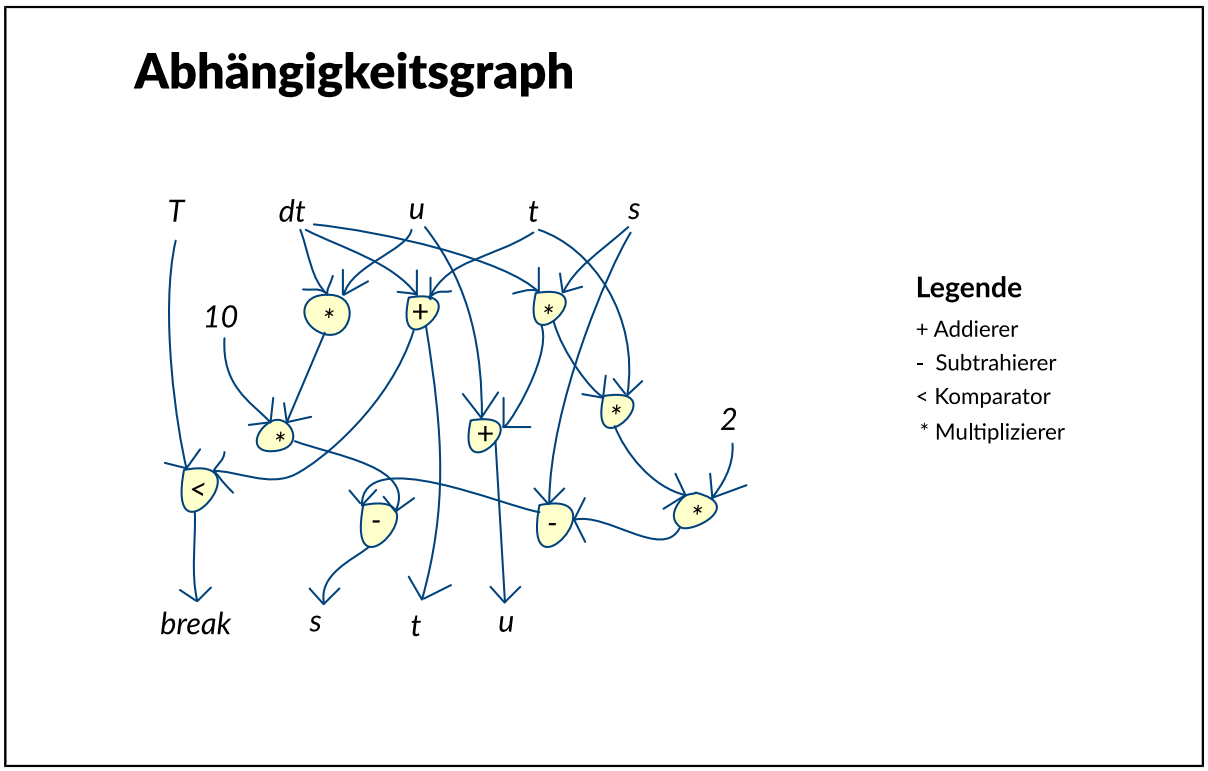
u = u + du = u + s*dt

s = s + ds = s + d(du / dt) = s + (d2u / dt)

mit "(d2u / dt) = -2t*du - 10u/dt" folgt

s = s - 2t*s*dt - 10*u*dt

Algorithmensynthese: Abhängigkeitsgraph



Die arithmetischen Operationen und ihre Abhängigkeiten voneinander stellen den Datenfluss eines Algorithmus dar. Datenabhängigkeiten entstehen dadurch, dass bestimmte Operationen zur Ausführung die Zwischenergebnisse vorangegangener Operationen benötigen. Aus den Datenabhängigkeiten ergibt sich somit eine Reihenfolge, in der die arithmetischen Operationen ausgeführt werden müssen.

Aus dem durch die Beschreibung in einer Hochsprache vorgegebenen Ablaufschema eines Algorithmus wird in der Algorithmensynthese zunächst ein so genannter Abhängigkeitsgraph extrahiert. In diesem Graphen sind alle Informationen über den Ablauf und die notwendigen arithmetischen Operationen enthalten. Der Graph enthält noch keine Informationen über den Zeitpunkt (Taktschritt), in dem eine Operation auszuführen ist. Es ist nur die extrahierte Reihenfolge der Operationen enthalten.

Während der Extraktion werden bereits Techniken zur Optimierung angewandt, um z. B. überflüssige Operationen im Algorithmus zu erkennen und zu entfernen. Hier kann auf bekannte Verfahren, die bei konventionellen Hochsprachenkompilern zur Anwendung kommen, zurückgegriffen werden. Ein typisches Beispiel für eine solche Optimierung ist das Zusammenführen von identischen Berechnungen und das Verändern der Reihenfolge von Anweisungen, um zum Beispiel ein Zwischenergebnis einer Berechnung mehrfach für folgende Berechnungen verwenden zu können. Dabei ist darauf zu achten, dass durch die Optimierungen das Ergebnis des Algorithmus nicht verändert wird.

Je nach Vorgabe des Entwicklers kann bereits zu diesem Zeitpunkt eine Anpassung der Algorithmusstruktur an die gewünschten Kriterien der resultierenden Schaltung vorgenommen werden. So kann zum Beispiel das Assoziativgesetz auf die arithmetischen Operationen angewendet werden, oder es können vorhandene Schleifen aufgelöst werden (loop unrolling).

Algorithmensynthese: Schleifen

Schleifen

Ggf. Auflösung von Schleifen möglich.
- Anzahl der Schleifendurchläufe ist datenabhängig.
- Anzahl der Schleifendurchläufe ergibt sich meist erst zur Laufzeit.
- Hoher Ressourcen-Bedarf

```
for i=1...n
  x[i]=0;
next i
```

```
x[1]=0;
x[2]=0;
.....
x[n]=0;
```

Verschachtelte Schleifen
- Werden von innen nach außen abgebildet.
- Innere Schleife wird als unteilbare Operation betrachtet.

```
for i=1...n
  for j=1...m
    x[i,j]=0;
  next j
next i
```

```
for i=1,n
  vx[i]=0;
next i
```

Für Schleifen kann sich eine Auflösung der Schleife und damit eine Parallelisierung der einzelnen Durchläufe anbieten, um die Ausführungsgeschwindigkeit des Algorithmus zu erhöhen. Da dies jedoch nur möglich ist, wenn die einzelnen Durchläufe voneinander datenunabhängig sind und sich die Anzahl der benötigten Schleifendurchläufe nicht erst zur Laufzeit des Algorithmus ergibt, kann diese Methode nur selten angewendet werden. Eine Parallelisierung aller Schleifen hätte auch einen sehr hohen Ressourcenbedarf der Schaltung zur Folge. Daher werden die meisten Schleifen auch in der Hardware sequenziell abgearbeitet. Bei der Abbildung ineinander verschachtelter Schleifen wird zuerst für die innerste Schleife ein Taktschema erstellt und dieses dann als eine einzelne, unteilbare Operation mit einer Laufzeit von n Taktschritten behandelt. Die in der Hierarchie nächst äußere Schleife benutzt diesen neu entstandenen Operator genau wie alle anderen Operatoren. Dieses Verfahren setzt sich so lange fort, bis die äußere Schleife abgebildet ist.

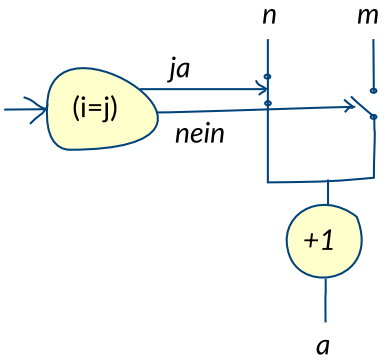
Algorithmensynthese: Bedingte Verzweigungen

Bedingte Verzweigungen

Abbildung bedingter Verzweigungen

→ "mutual exclusion" nutzen
Beispiel: Jeder Zweig eine Addition
→ Nur ein Addierer notwendig

```
if (i==j)
    a = n+1
else
    a = m+1
```



Bei der Abbildung einer bedingten Verzweigung kann zur Ressourcenminimierung ausgenutzt werden, dass die Zweige niemals gleichzeitig ausgeführt werden (mutual exclusion). So kann für den Fall, dass in jedem Zweig eine Addition ausgeführt werden muss, darauf verzichtet werden, zwei Addierer zu implementieren. Es reicht ein Addierer, der durch den Steuerpfad für die arithmetischen Operationen des jeweils aktiven Zweigs verwendet wird.

Algorithmensynthese: Schritte der Algorithmensynthese

Schritte der Algorithmensynthese

Abbildung des Abhängigkeitsgraphen
→ Zwei Dimensionen:
- Taktschema
- Hardwareressourcen

Abbildung auf Taktschema (Ablaufplanung, Scheduling)
→ Festlegung, welche Operation in welchem Taktschritt ausgeführt wird

Abbildung auf Hardwareressourcen
→ Festlegung, welche Operation von welcher Ressource durchgeführt wird
→ Zwei Schritte:
- Ressourcenreservierung (resource allocation)
- Ressourcenzuordnung (resource assignment)

Um den Algorithmus in eine Schaltung zu überführen, ist es nötig, den Abhängigkeitsgraphen in zwei Dimensionen abzubilden:

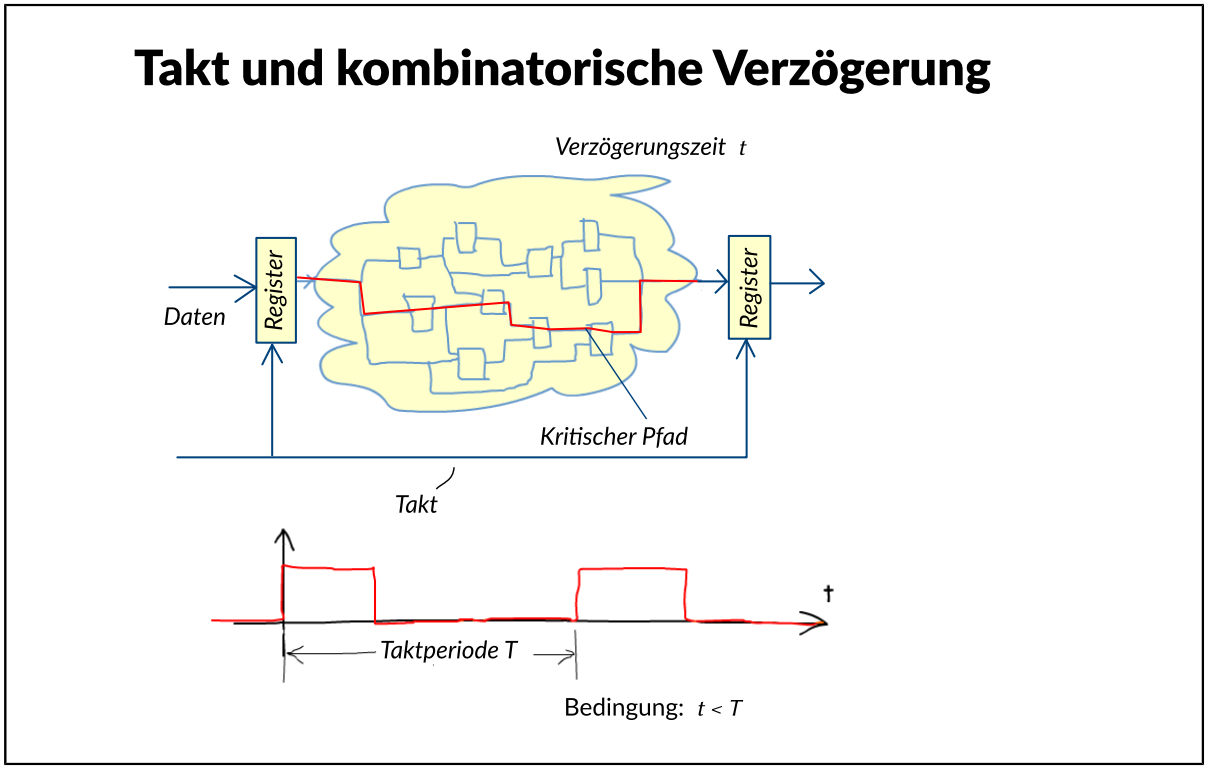
- Taktschema (Performanceoptimierung)
- Hardwareressourcen (Kostenoptimierung)

Die Abbildung auf ein Taktschema, das so genannte Scheduling, dient der Erstellung eines Zeitplans zur Ausführung des Algorithmus. Die Abbildung auf die vorhandenen Hardwareressourcen legt fest, welche Operation von welcher Ressource ausgeführt wird. Dieses geschieht in zwei Schritten.

- Ressourcenreservierung (resource allocation)
- Ressourcenzuordnung (resource assignment)

Die drei Schritte Scheduling, Resource Allocation und Resource Assignment können nicht als unabhängige Teilaufgaben durchgeführt werden, da sie sich gegenseitig stark beeinflussen. So ist die Erstellung eines Taktschemas nur möglich, wenn bereits Ressourcen reserviert sind. Wie viele Ressourcen zu reservieren sind, hängt jedoch vom erstellten Taktschema ab.

Algorithmensynthese: Takt und kombinatorische Verzögerung



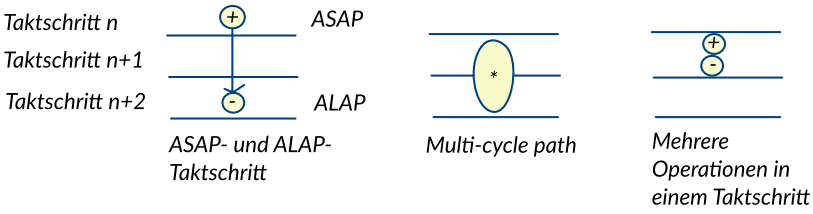
Bei synchronen Schaltungen, die hier ausschließlich behandelt werden, übernehmen speichernde Elemente wie Register oder Flip-Flops Eingangsdaten zu einem definierten Zeitpunkt, der durch einen Takt (z. B. seine positive Flanke) vorgegeben ist. Zwischen zwei Taktschritten werden die Daten in einer kombinatorischen Logik verarbeitet. Die dabei auftretende Gesamtverzögerungszeit t (kombinatorische Verzögerung) muss grundsätzlich kleiner sein als die Taktperiode T.

Der kritische Pfad ist der längste Pfad, den ein Signal durch einen kombinatorischen Schaltungsteil zwischen zwei speichernden Elementen innerhalb einer Taktperiode zurückzulegen hat. Die Länge des Pfads wird durch die Anzahl der zu durchlaufenden logischen Verknüpfungen eines Signals bestimmt, d.h.die Gesamtverzögerungszeit ergibt sich als Summe der Gatter- und Leitungsverzögerungszeiten auf dem Pfad. Der kritische Pfad bestimmt somit die maximal erreichbare Taktfrequenz einer Schaltung.

Algorithmensynthese: Scheduling – Erstellung eines Taktschemas

Scheduling - Erstellung eines Taktschemas

- Taktschema: Verteilung der Operationen auf die Taktschritte.
- Taktschema ist abhängig von Hardwareressourcen.
- Für Operationen jenseits des längsten Pfades:
  - ASAP-Taktschritt (As Soon As Possible)
  - ALAP-Taktschritt (As Late As Possible)
- "Multi-cycle path": Mehrere Taktschritte für eine Operation
- Mehrere Operationen in einem Taktschritt sind möglich



ASAP- und ALAP-Taktschritt

Multi-cycle path

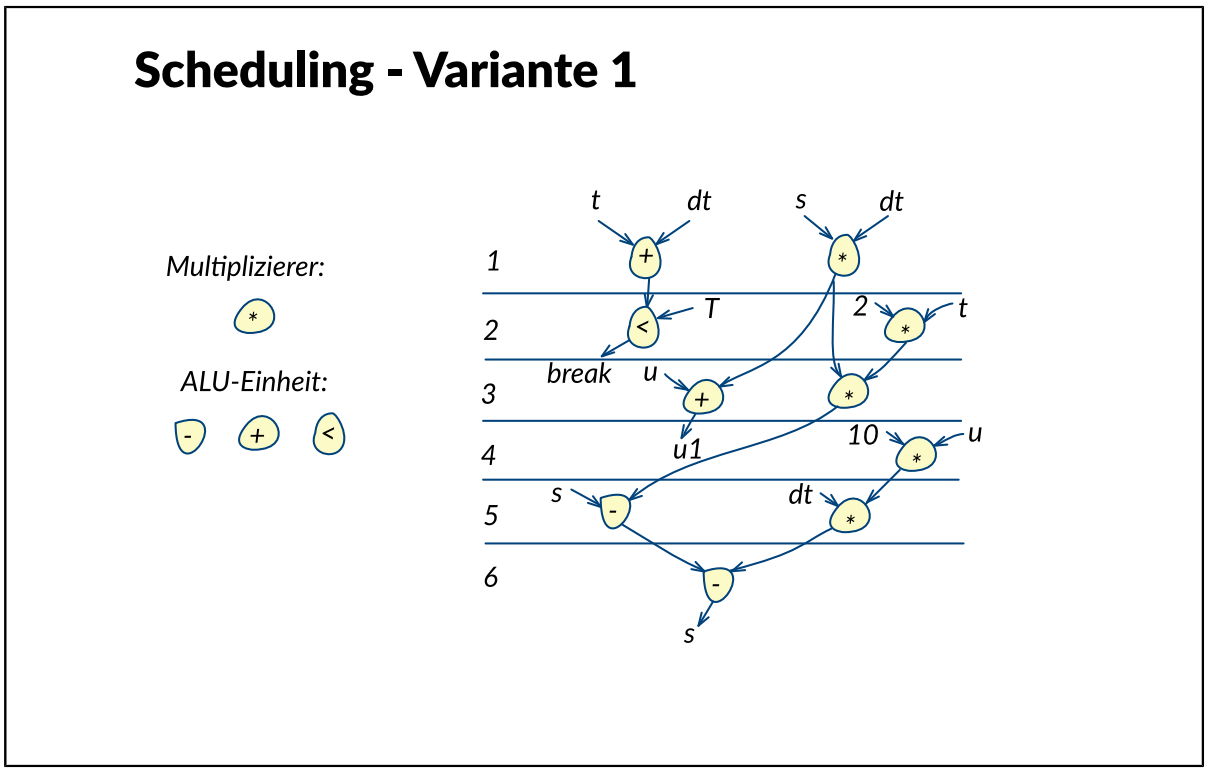
Mehrere Operationen in einem Taktschritt

Mit der Erstellung eines Taktschemas erfolgt die Festlegung, in welchem Taktschritt die einzelnen Operationen des Algorithmus ausgeführt werden. Das Taktschema ist dabei erheblich von der Anzahl der vorhandenen Hardwareressourcen abhängig. So müssen alle Operationen des Algorithmus zwangsläufig sequenziell abgearbeitet werden, wenn nur eine Ressource zur Ausführung von arithmetischen Operationen zur Verfügung steht. Sind jedoch Ressourcen in unbegrenzter Anzahl vorhanden, kann die Ausführung bis zum theoretisch möglichen Grad, der durch die Struktur des Algorithmus vorgegeben wird, parallelisiert werden.

Ist die Anzahl der zur Verfügung stehenden Ressourcen mindestens so groß wie die Anzahl der benötigten, entspricht die minimale Anzahl benötigter Taktschritte dem längsten Pfad des abzubildenden Algorithmus. Für die Operationen, die nicht auf diesem Pfad liegen, entsteht dabei ein Zeitfenster, welches den frühestmöglichen und spätestmöglichen Taktschritt angibt, in dem die Operation ausgeführt werden kann. Der früheste Taktschritt ist dabei der so genannte ASAP-Taktschritt (As Soon As Possible) und der späteste der so genannte ALAP-Taktschritt (As Late As Possible).

Wird für die Ausführung einer Operation mehr als ein Taktschritt benötigt, besteht die Möglichkeit, die Dauer dieser Operationen im Datenpfad auf mehrere Taktschritte des Steuerpfades auszudehnen. Der kombinatorische Pfad wird dabei nicht durch Register unterbrochen, sondern es wird bewusst akzeptiert, dass die kombinatorische Verzögerung in diesem Schaltungsteil größer ist als eine Taktperiode. Der so entstehende Pfad wird "multi-cycle path" genannt. Die Steuerung des Datenpfades wird in diesem Fall so angepasst, dass das Ergebnis des multi-cycle path erst nach der minimal notwendigen Anzahl von Taktschritten erwartet wird. Andererseits ist es möglich, mehrere Operationen sequenziell innerhalb eines Taktschrittes auszuführen, sofern die Summe aller Verzögerungen die Länge eines Taktschrittes nicht überschreitet.

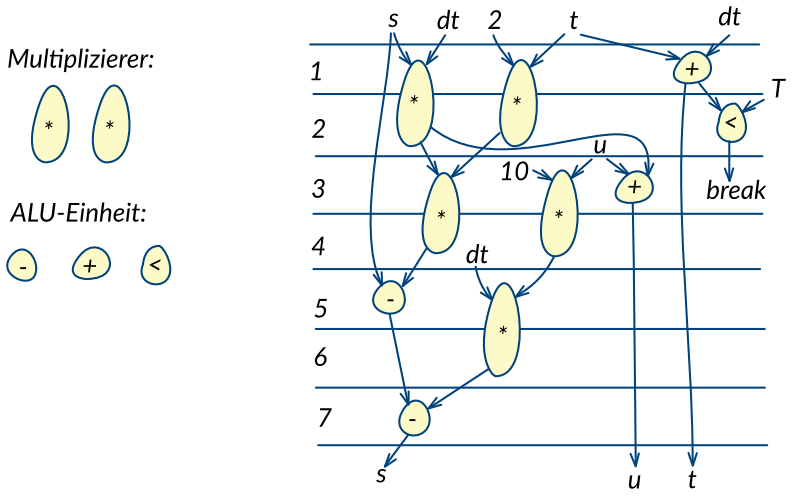
Algorithmensynthese: Scheduling Variante 1



Als erste Scheduling-Variante ist hier eine Realisierung dargestellt mit einem schnellen, d.h. in einem Taktschritt arbeitenden Multiplizierer und einer ALU, die Additionen, Subtraktionen sowie Vergleiche ausführen kann. Mit diesen Ressourcen ist eine Abarbeitung des Algorithmus in 6 Taktschritten möglich.

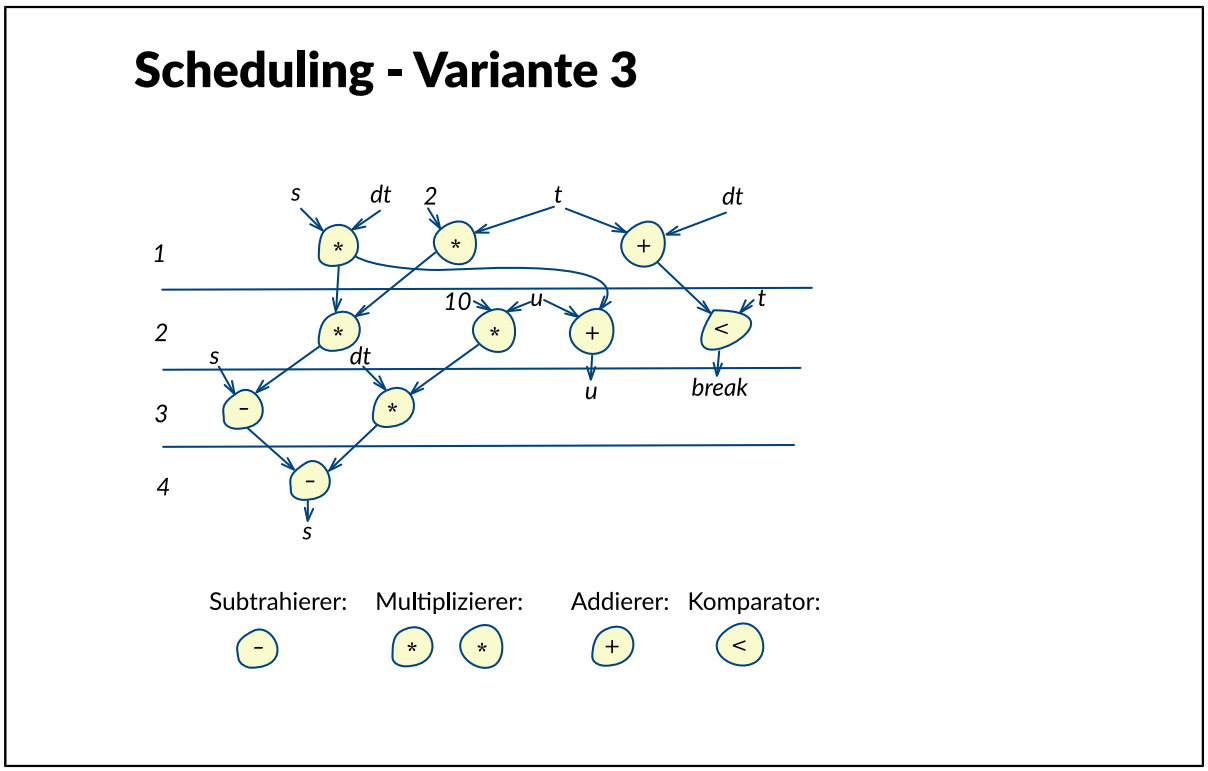
Algorithmensynthese: Scheduling Variante 2

Scheduling - Variante 2



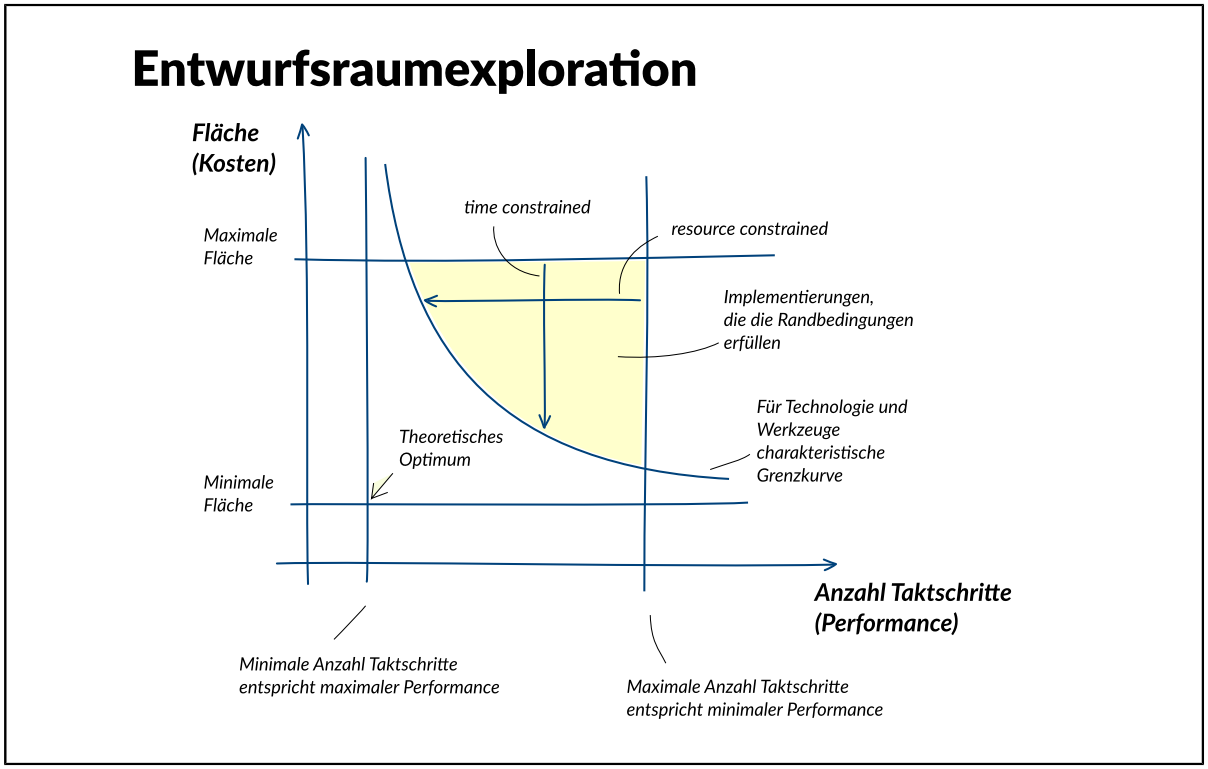
Als zweite Scheduling-Variante ist hier eine Realisierung mit zwei langsamen Multiplizierern dargestellt. Obwohl die beiden Multiplizierer parallel arbeiten können, sind für die Abarbeitung des Algorithmus 7 Taktschritte nötig.

Algorithmensynthese: Scheduling Variante 3



Hier ist eine Variante dargestellt, die zwei schnelle Multiplizierer, einen Addierer, einen Subtrahierer und einen Komparator verwendet. Damit ist eine Abarbeitung des Algorithmus in 4 Taktschritten möglich.

Algorithmensynthese: Entwurfsraumexploration



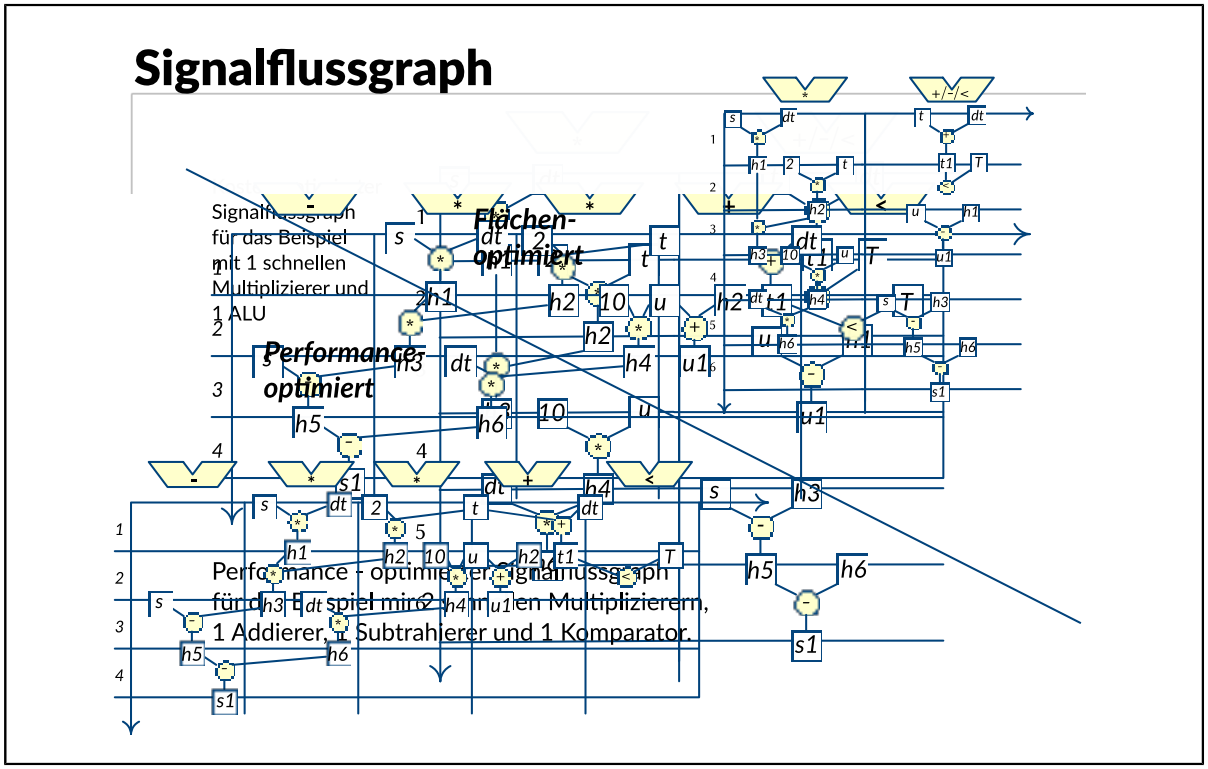
In der Abbildung wird der durch Fläche (Kosten) und Anzahl Taktschritte (Performance) beschriebene Entwurfsraum dargestellt. Die Grenzen, die durch die Spezifikation oder durch technologieabhäningige Parameter vorgegeben sind, sind in der Abbildung eingezeichnet.

Bei der Erstellung eines Taktschemas kann von zwei unterschiedlichen Randbedingungen ausgegangen werden:

- Anzahl der Ressourcen und damit die Fläche ist festgelegt (resource constrained)
- Anzahl der Taktschritte und damit die Performance ist festgelegt (time constrained)

Im Falle einer Festlegung der Anzahl der Ressourcen, ist es das Ziel des Scheduling, den Algorithmus mit den zur Verfügung stehenden Ressourcen in möglichst wenig Taktschritten abzuarbeiten.
Falls die Anzahl der Taktschritte festgelegt wird, soll der Algorithmus mit einer möglichst geringen Anzahl von Ressourcen innerhalb der vorgegebenen Anzahl von Taktschritten ausgeführt werden.
Diese beiden Fälle sind Teil des allgemeinen Optimierungsproblems, welches im Rahmen der Algorithmensynthese zu lösen ist. Das Optimierungsziel sind die beiden gegensätzlichen Kriterien minimaler Flächenbedarf und maximale Performance der Schaltung. Ein minimaler Flächenbedarf entspricht dabei einem minimalen Ressourcenbedarf, da jede zusätzlich verwendete Ressource auch zusätzliche Chipfläche beansprucht. Während der Optimierung ist darauf zu achten, dass die vorgegebenen Randbedingungen (constraints) eingehalten werden. Die Abbildung zeigt den durch Randbedingungen und Technologiegrenzen festgelegten gültigen Bereich im Entwurfsraum.

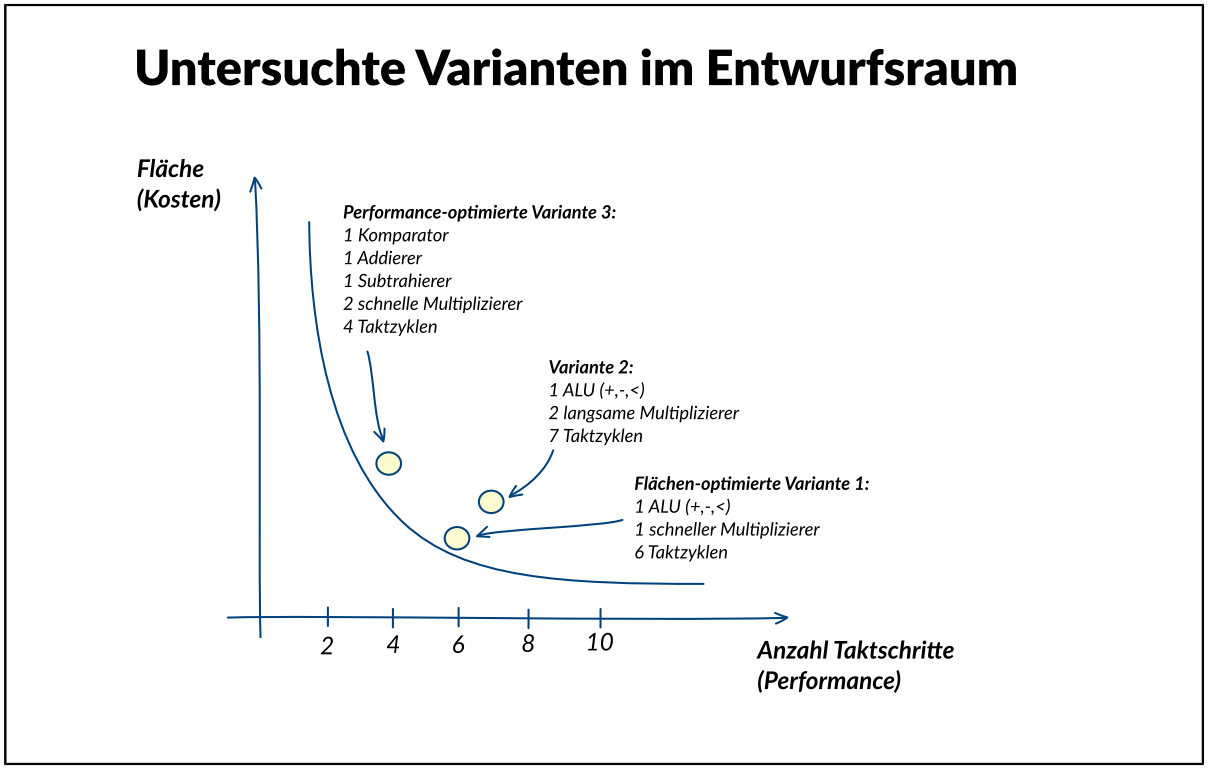
Algorithmensynthese: Signalflussgraph



Ein Signalflussgraph enthält alle notwendigen Informationen, um eine Schaltung auf RT-Ebene zu generieren. Er legt fest, welche Operation in welchem Taktschritt mit welcher Hardwareressource ausgeführt wird. Im Signalflussgraphen ist von oben nach unten die Abfolge der Taktschritte zu sehen. An den Übergängen von einem Takt zum nächsten sind die Register angeordnet, da sie die Speicherung von Daten über den aktuellen Taktzyklus hinaus übernehmen.

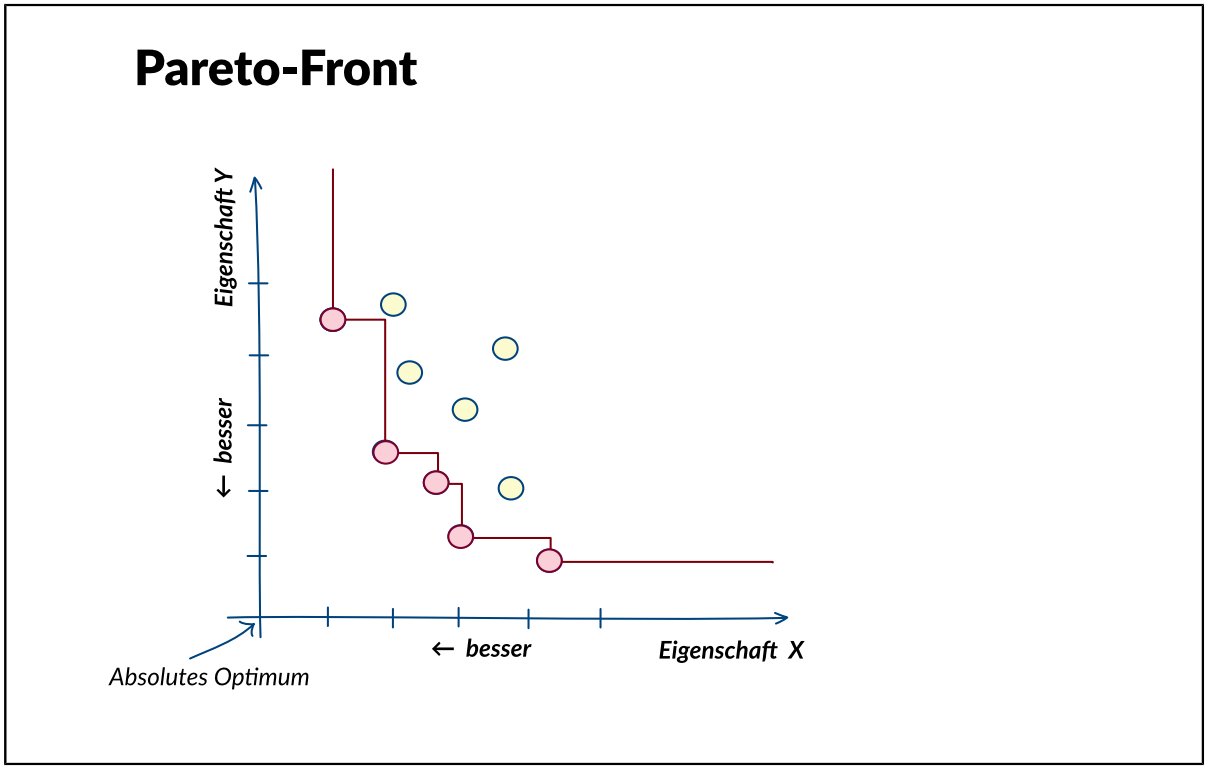
Die beiden Abbildungen zeigen noch einmal die Signalflussgraphen des Beispielalgorithmus mit unterschiedlichen Optimierungszielen. Beim Kosten-optimierten Signalflussgraphen erfolgte eine stärkere Reduktion des Ressourcenbedarfs, während beim Performance-optimierten Signalflussgraphen die Geschwindigkeit auf Kosten eines größeren Ressourcenbedarfs erhöht wurde.

Algorithmensynthese: Varianten im Entwurfsraum



Im Entwurfsraum ergibt sich für die untersuchten Varianten die gezeigte Anordnung. Unter der Bedingung , dass ein langsamer Multiplizierer zwar flächengünstiger ist als ein schneller, zwei langsame jedoch mehr Fläche benötigen als ein schneller, ergibt sich die Variante 1 als flächengünstigste.

Algorithmensynthese: Pareto-Front



Als gute Lösungen kommen nur Varianten in betracht, die auf einer sogenannten Pareto-Front liegen. Diese Front ist wie folgt definiert:

Alle Punkte auf der Pareto-Front sind solche, für die es nicht möglich ist, Punkte mit einer verbesserten Eigenschaft zu finden, ohne dass zugleich eine andere Eigenschaft verschlechtert werden müsste.

Algorithmensynthese: Ressourcenreservierung und Ressourcenzuordnung

Ressourcenreservierung und Ressourcenzuordnung

Ressourcenreservierung (resource allocation)

-> Die Anzahl ergibt sich aus Taktschema.
-> Es werden so viele Instanzen benötigt, wie parallel in einem Takt sein müssen.
-> Zur Speicherung der Zwischenergebnisse werden Register benötigt (Wiederverwendung möglich).

Ressourcenzuordnung (resource assignment)

-> Ressourcen sollen möglichst gleichmäßig ausgelastet werden.
-> Anzahl Multiplexer
- abhängig von Datenstrom
- in der Regel ein Multiplexer pro Makrozelle/Register.

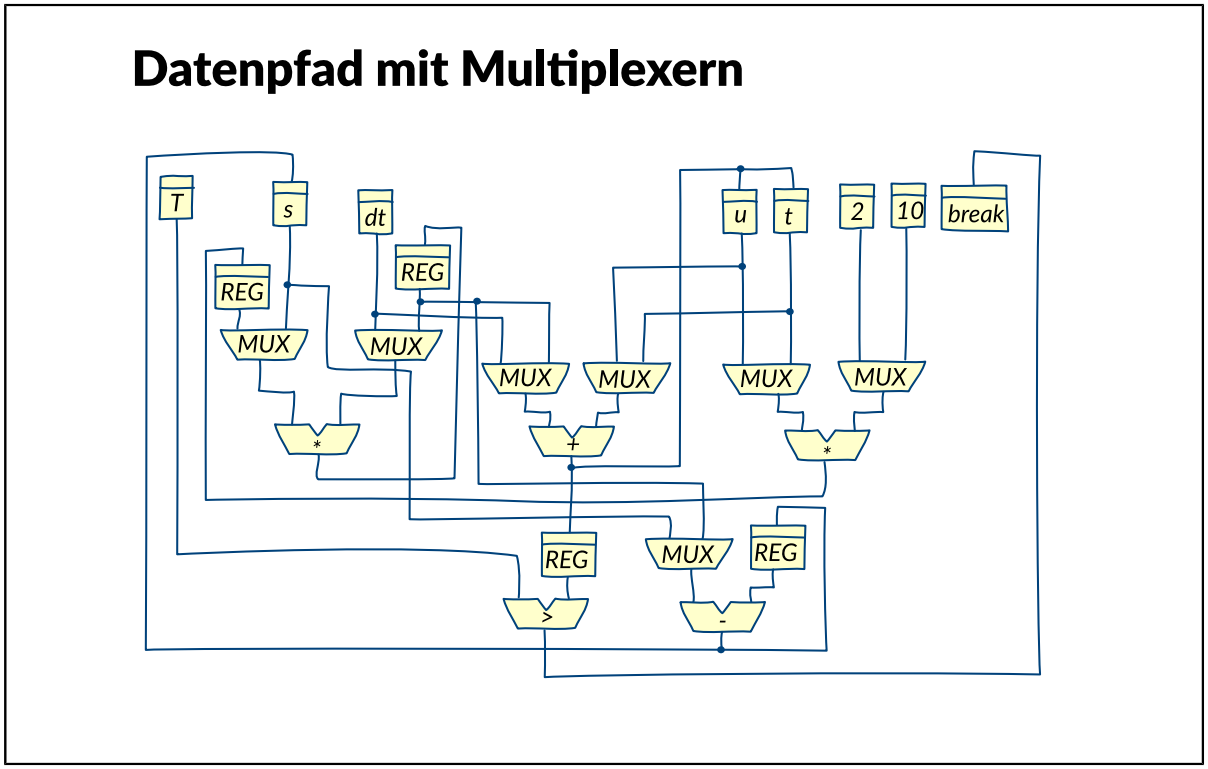
-> Anzahl Register
- ergibt sich aus maximaler Anzahl, der in einem Taktschritt zu speichernden Daten.

Bei der Ressourcenreservierung werden die Ressourcen instanziiert, die während der Erstellung des Taktschemas verplant wurden. Die Anzahl der einzelnen Ressourcen ergibt sich dabei direkt aus dem Taktschema. Von einem bestimmten Ressourcentyp werden genau so viele Einheiten benötigt, wie maximal in einem Taktschritt parallel aktiv sein müssen. Bei der Ressourcenzuordnung werden den einzelnen Operationen die instanziierten Ressourcen zugeordnet.

Für alle Daten, deren Generierung und Weiterverarbeitung in unterschiedlichen Taktschritten erfolgt, werden Register zur Speicherung benötigt. Die Anzahl der Register ergibt sich dabei aus der maximalen Anzahl gleichzeitig in einem Taktschritt zu speichernden Daten. Durch die Mehrfachverwendung von Registern für Daten, die ausschließlich in unterschiedlichen Taktschritten gültig sind, können Ressourcen eingespart werden. Das jeweils abzuspeichernde Datum wird in diesem Fall durch einen dem Registereingang vorangeschalteten Multiplexer selektiert. Die Dauer, die ein Datum in einem Register gespeichert werden muss, wird durch den ermittelten Ablaufplan des Algorithmus vorgegeben.

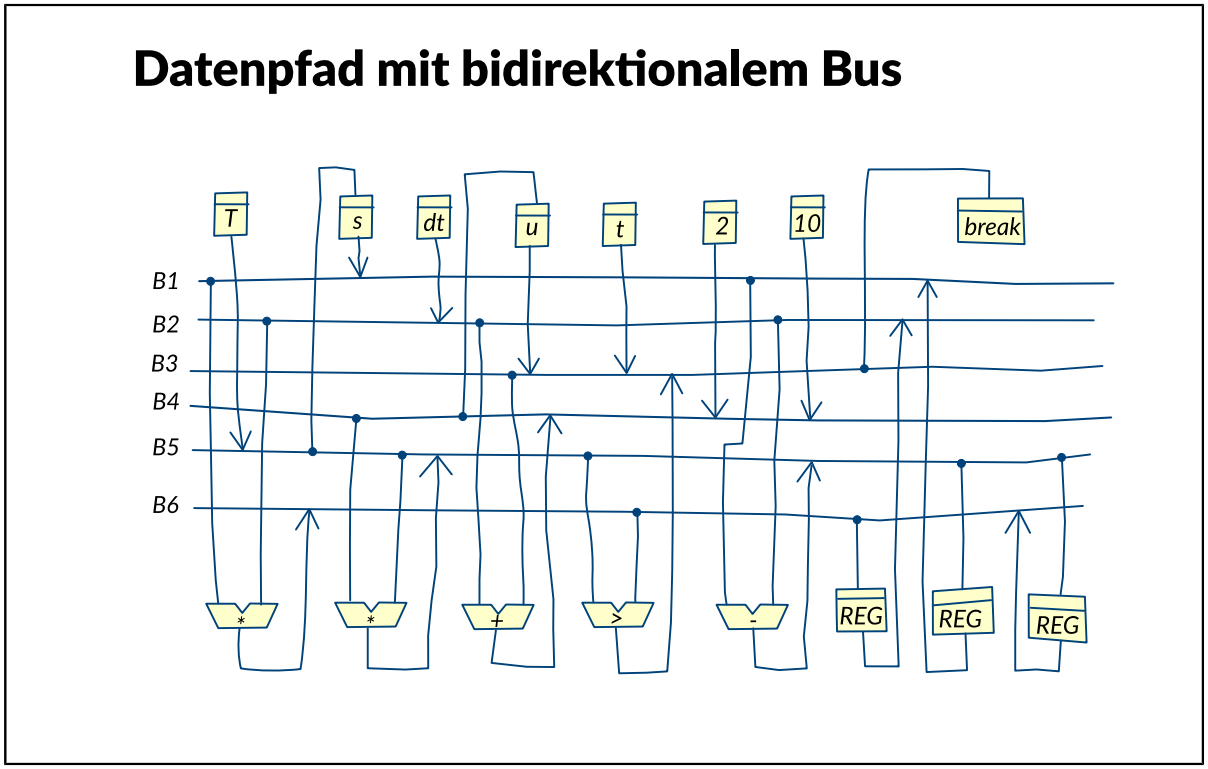
Wenn alle notwendigen Ressourcen instanziiert sind, müssen diese noch den einzelnen Operationen zugeordnet werden. Die Zuordnung sollte dabei so erfolgen, dass ein möglichst optimaler Datenfluss durch den Datenpfad gewährleistet ist und die Anzahl bzw. Breite der Multiplexer an den Blockeingängen minimal wird. Für die Minimierung der Multiplexerressourcen sollten die arithmetischen Makrozellen möglichst gleichmäßig ausgelastet werden, da eine Zuordnung vieler Operationen zu nur einer Ressource zu sehr komplexen und damit langsamen Multiplexerstrukturen führen würde.

Algorithmensynthese: Datenpfad mit Multiplexern



Aus dem Signalflussgraphen lassen sich die Struktur des Datenpfads und das Zustandsübergangsdiagramm des Steuerpfads generieren. Die Struktur des Datenpfads kann dabei aus der Struktur des Signalflussgraphen abgeleitet werden, indem für jeden Taktschritt bestimmt wird, welche Daten einem Block zugeführt werden müssen. Die Anzahl aller Möglichkeiten ergibt dabei die notwendige Anzahl der Eingänge des Eingangsmultiplexers. Alle in Frage kommenden Makrozellen, die eine Datenquelle für die auszuführenden Operationen sein können, müssen mit den Multiplexern verbunden werden. Datenquelle und Datensenke sind sowohl Register als auch Makrozellen, die arithmetische Operationen ausführen. Der sich daraus ergebende Datenpfad für das hier verwendete Beispiel in der Variante 3 ist in der Abbildung dargestellt.

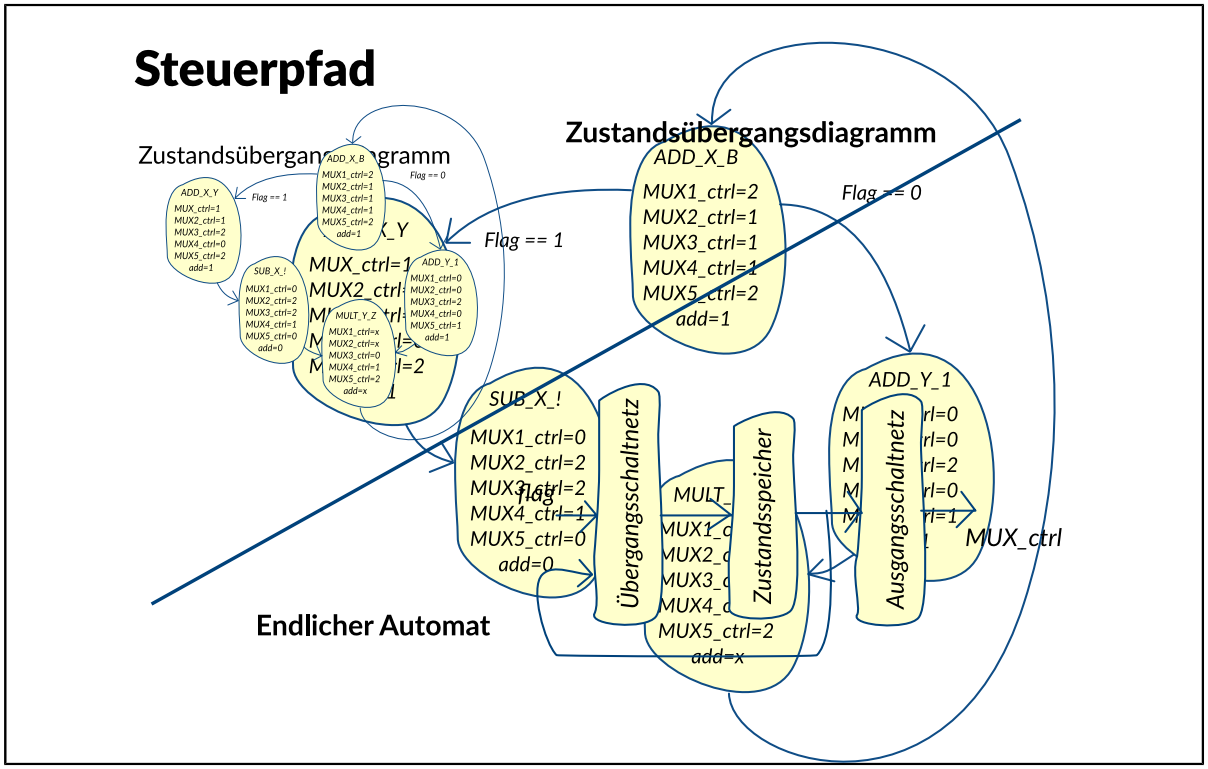
Algorithmensynthese: Datenpfad mit bidirektionalem Bus



Als Alternative zu einer Realisierung mit Multiplexern kann der Datenpfad auch mit einem bidirektionalen Bus zum Datentransfer zwischen den arithmetischen Makrozellen aufgebaut werden. Alle Makrozellen lesen und schreiben dabei auf einen gemeinsamen Datenbus. Über diesen werden die einzelnen Operatoren zu den Verarbeitungseinheiten transferiert. Das Bild zeigt exemplarisch den Aufbau eines Datenpfads der Variante 3 des Beispiels mit einem bidirektionalen Bus.

Busse sind zwar in der Regel kostengünstiger zu implementieren, da auf die (teuren) Multiplexer verzichtet werden kann, sie sind aber auch wesentlich langsamer als Multiplexerstrukturen.

Algorithmensynthese: Steuerpfad



Zur Beschreibung des Steuerpfads wird aus dem Signalflussgraphen ein so genanntes Zustandsübergangsdiagramm erstellt, welches einen endlichen Automaten beschreibt. Das Zustandsübergangsdiagramm lässt sich ebenfalls in Form eines Graphen darstellen. Die Knoten sind dabei die Zustände und die gerichteten Kanten die Zustandsübergänge. Zu jedem Zustandsübergang gehört ein Eingangsbitmuster, das den Zustandswechsel auslöst. Das resultierende Schaltnetz, welches in Abhängigkeit von den Eingangssignalen und dem aktuellen Zustand den Folgezustand bestimmt, wird Übergangsschaltnetz genannt. Zur Berechnung der Ausgänge des endlichen Automaten dient das so genannte Ausgangsschaltnetz. Wenn das Ausgangsschaltnetz die Ausgänge nur in Abhängigkeit vom aktuellen Zustand berechnet, wird der endliche Automat Moore-Automat genannt. Um einen so genannten Mealy-Automaten handelt es sich, wenn die Ausgänge eine Funktion des aktuellen Zustands und der Eingänge sind. Die Abbildung zeigt ein exemplarisches Zustandsübergangsdiagramm für eine Datenpfadrealisierung mit Multiplexern. Das Zustandsübergangsdiagramm enthält eine bedingte Verzweigung, deren Bedingung ein gesetztes bzw. nicht gesetztes Flag ist. Es handelt sich um einen Moore-Automaten.

Algorithmensynthese: Nachoptimierungsmöglichkeiten

Nachoptimierungsmöglichkeiten

1) Zustandsminimierung
-> Sequenzen übereinstimmender Zustände ermitteln und zusammenfassen.

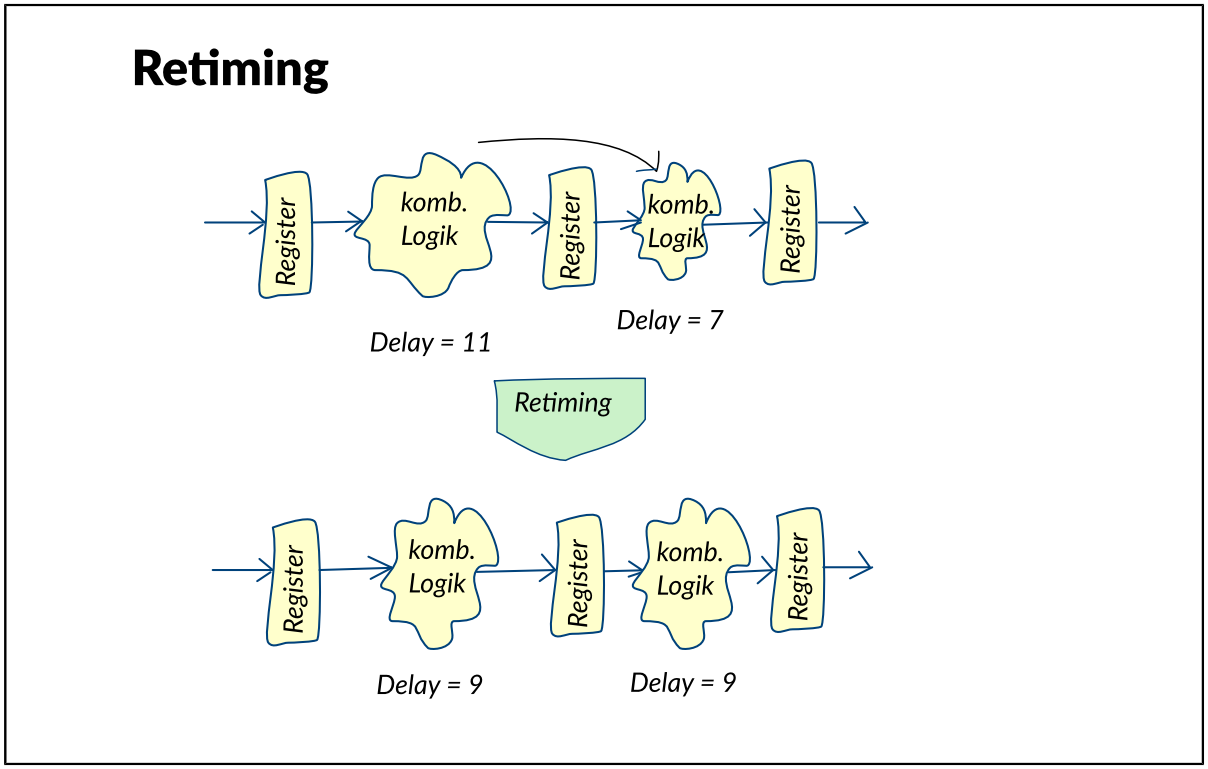
2) Retiming
-> Kombinatorische Schaltungsteile auf die zur Verfügung stehenden Taktzyklen möglichst gleichmäßig verteilen.

3) Pipelining
-> Komplexe Operationen in Teiloperationen (= Pipeline-Stufen) aufteilen.

Eine Reduzierung der notwendigen Ressourcen im Steuerpfad ist durch eine Zustandsminimierung des endlichen Automaten zu erreichen. Dabei wird das Zustandsübergangsdiagramm des endlichen Automaten auf Sequenzen von übereinstimmenden Zuständen untersucht. Zustände stimmen dann überein, wenn sie die gleichen Werte an den Ausgängen erzeugen und ihre Folgezustände identisch sind. In einem solchen Fall lassen sich die Zustände zusammenfassen.

Um die Schaltung auf eine möglichst hohe Taktfrequenz hin zu optimieren, kommt das so genannte Retiming zur Anwendung. Beim Retiming werden die kombinatorischen Schaltungsteile möglichst gleichmäßig auf die zur Verfügung stehenden Taktschritte verteilt. Dadurch wird die Länge des kritischen Pfads der Schaltung minimiert.

Algorithmensynthese: Retiming



Falls eine gleichmäßige Verteilung der kombinatorischen Blöcke einer Schaltung nicht ausreicht, um die gewünschte Taktfrequenz zu erreichen, müssen zusätzliche Register eingefügt werden. Dabei werden kombinatorische Blöcke der Schaltung, deren kritischer Pfad für die geforderte Taktfrequenz zu lang ist, durch Register in kleinere Blöcke unterteilt. Dadurch verkürzt sich der kritische Pfad und die maximal erreichbare Taktfrequenz steigt an. Auf Grund der eingefügten Register erhöht sich die Latenzzeit der Schaltung. Es werden also mehr Taktschritte für die gleiche Aufgabe benötigt. Durch die mögliche Erhöhung der Taktfrequenz kann dieser Geschwindigkeitsverlust aber wieder kompensiert werden.

Eine weitere geschwindigkeitssteigernde Maßnahme stellt das Pipelining dar. Hier wird in jedem Taktschritt immer nur jeweils ein Teil der gesamten Operation abgearbeitet. Das Ergebnis dieser Teilaufgabe wird in Registern gespeichert und im nächsten Taktschritt weiterverarbeitet. Die Teile einer Schaltung, die die einzelnen Teilaufgabe ausführen, werden Pipelinestufen genannt. Hat eine Teiloperation die Pipelinestufe verlassen, kann aus der vorhergehenden Stufe die nächste Operation übernommen und verarbeitet werden. Dies führt im Idealfall dazu, dass in jedem Taktschritt eine neue Operation in die aus den einzelnen Stufen zusammengesetzte Pipeline eingespeist werden kann. So kann trotz einer Verarbeitungszeit von mehreren Taktschritten für eine Operation, mit Pipelining die Gesamtbearbeitungszeit der Operation verringert werden. Ein Nachteil des Pipelinings ist jedoch der erhöhte Ressourcenbedarf der Schaltung, da jede Pipelinestufe ihre eigenen Ressourcen zur Verarbeitung der Operationen benötigt. Eine Mehrfachnutzung ist nur bedingt möglich.