

# Synthese Eingebetteter Systeme

Wintersemester 2012/13

## 9 – Synthese: Grundlagen

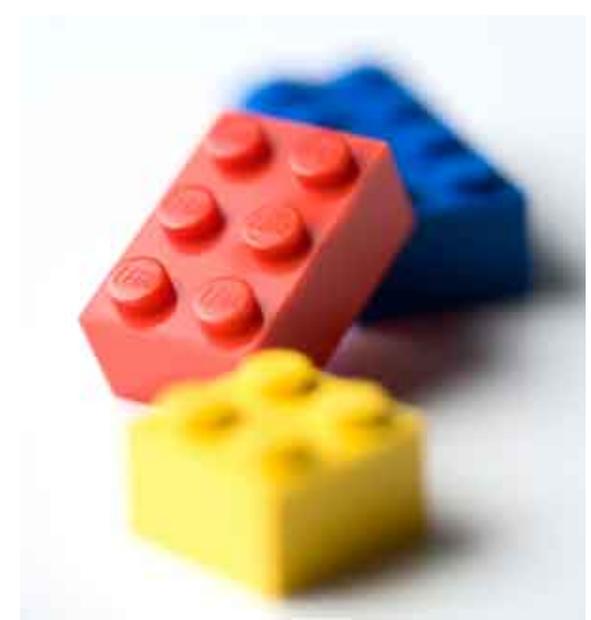
Michael Engel  
Informatik 12  
TU Dortmund

2012/11/26

# Synthese: Grundlagen

---

- Überblick: Synthese
- High-Level-Synthese
- Werkzeugfluss
- Übersetzung und Modellierung
  - Kontroll-/Datenfluss
- Allokation
- Scheduling
- Bindung
- Generierung
- Ausgabemodell



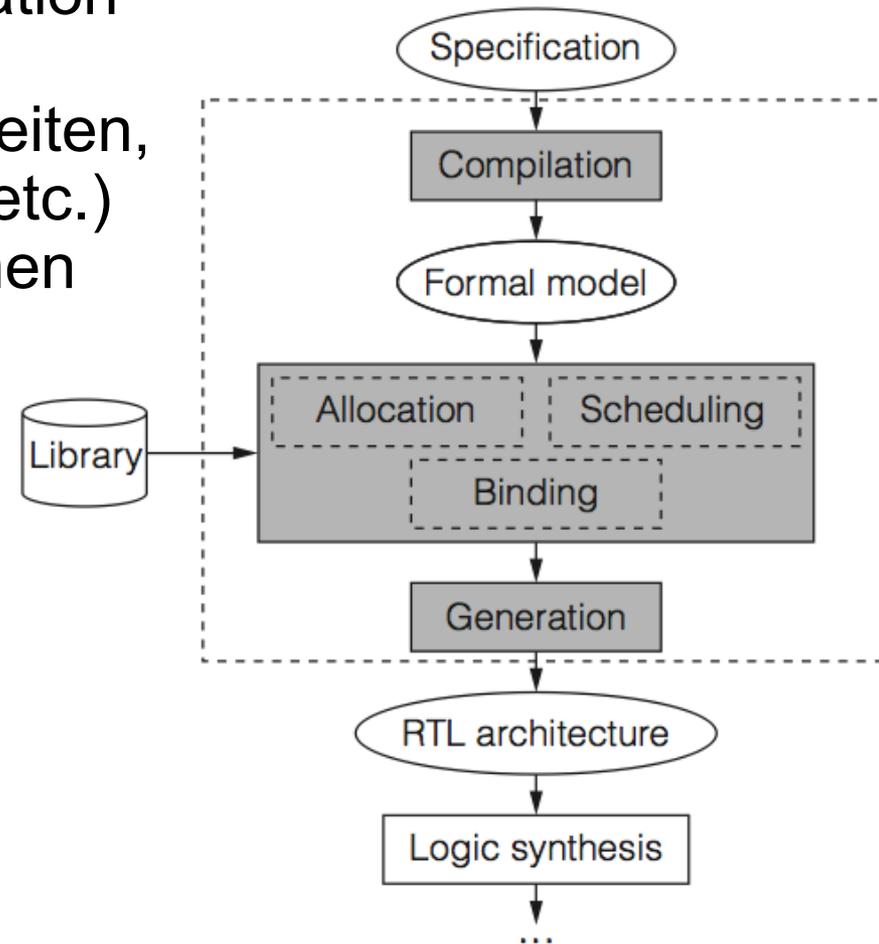
# Synthese

---

- Definitionen (Wdh.):
  - „**Design synthesis can be defined as the transformation of a design to a level of lower abstraction.**“
- Präzisierung:
  - „Transformation of a design from a point in the functional domain to one in the structural domain“
- **High-Level-Synthese (HLS)**
- **Logiksynthese**
  - „maps RTL-descriptions onto a specific structure suitable for the target architecture: **low-level synthesis**“

# Werkzeugfluss der High-Level-Synthese

1. **Übersetzung** der Spezifikation
2. **Allokation** der Hardware-ressourcen (Funktionseinheiten, Speicherelemente, Busse etc.)
3. **Scheduling** von Operationen auf Taktzyklen
4. **Bindung** von Operationen an Funktionseinheiten
5. **Bindung** von Variablen an Speicherelemente
6. **Bindung** von Transfers an Busse
7. **Generierung** der RTL-Architektur



# Übersetzung und Modellierung

---

- Erster Schritt der HLS: Übersetzung der Funktionsbeschreibung
  - Übersetzt eingegebene Beschreibung in formale Darstellung
- Beinhaltet verschiedene Codeoptimierungen
  - dead-code elimination
  - false data dependency elimination
  - constant folding and loop transformations
- Erzeugtes formales Modell zeigt Daten- und Kontrollflussabhängigkeiten zwischen den Operationen

# Übersetzung und Modellierung

- Datenabhängigkeiten dargestellt als Datenflussgraph (DFG)
  - Jeder Knoten stellt eine Operation dar
  - Kanten stehen für Eingabe-, Ausgabe- und temporäre Variablen
- Beispiel:

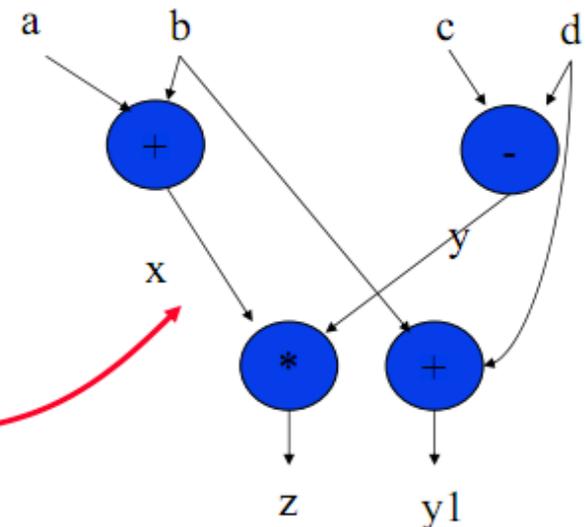
**gegebener Basisblock:**

```
x = a + b;  
y = c - d;  
z = x * y;  
y = b + d;
```

**Single Assignment Form:**

```
x = a + b;  
y = c - d;  
z = x * y;  
y1 = b + d;
```

**Datenflussgraph**



# Datenflussgraphen

---

- Reine DFG modellieren nur Datenabhängigkeiten
- Oft durch Entfernen von Kontrollflussabhängigkeiten der Ausgangsspezifikation erzeugbar:
  - Vollständiges Abrollen von Schleifen
  - Auflösung bedingter Zuweisungen durch Wertmultiplex
- Resultierender DFG zeigt explizit in Spezifikation enthaltene Parallelität
  - Kann zu großen Darstellungen mit hohem Speicherbedarf zur Synthesezeit führen
  - Unterstützt keine *unbounded loops* oder goto-Statements
- Verwendung reiner DFG daher nur für wenige Anwendungen sinnvoll

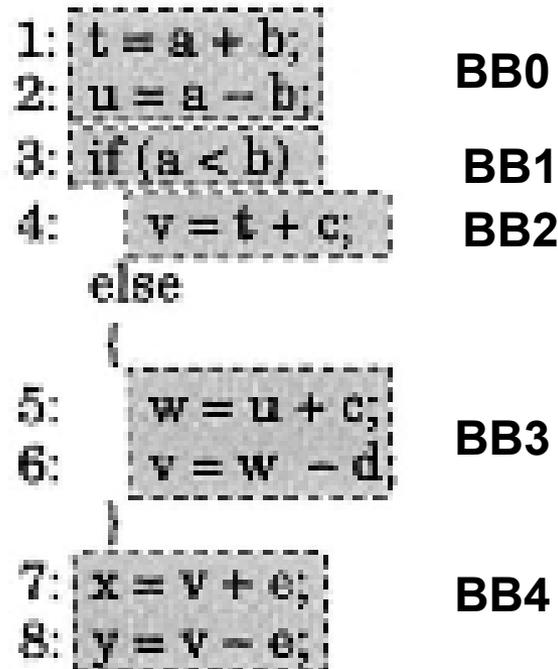
# Kontroll-/Datenflussgraphen: CDFG

---

- CDFG: Erweiterung des DFG-Modells durch Kontrollflussabhängigkeiten
- Gerichteter Graph
  - Kanten = Kontrollfluss
  - Knoten = Basisblöcke
- Kanten können bedingt sein
  - Darstellung von if- und switch-Konstrukten
- CDFG zeigt Datenabhängigkeiten innerhalb von Basisblöcken und stellt Kontrollfluss zwischen den Basisblöcken dar

# Basisblöcke

- **Basisblock:** Folge von Anweisungen, die weder Sprünge noch interne Sprungziele oder Aussprungpunkte (z.B. return-Anweisungen) enthält



# Beispiel: CDFG

Code –  
Basisblöcke

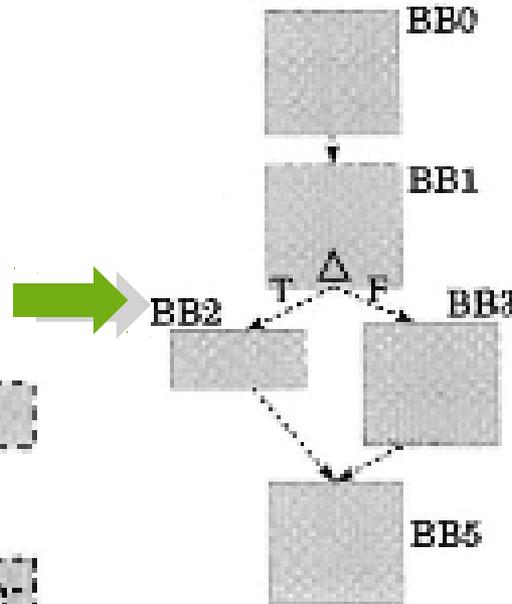
BB0  
1: `t = a + b;`  
2: `u = a - b;`

BB1  
3: `if (a < b)`

BB2  
4: `v = t + c;`  
else

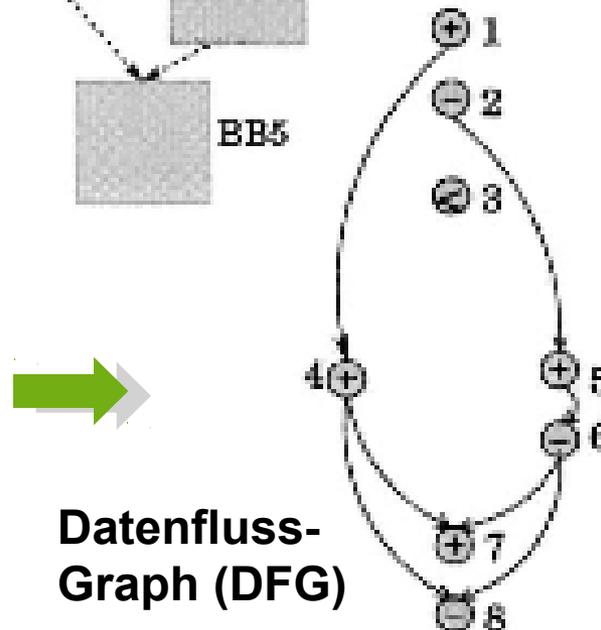
BB3  
5: `w = u + c;`  
6: `v = w - d;`

BB4  
7: `x = v + e;`  
8: `y = v - e;`

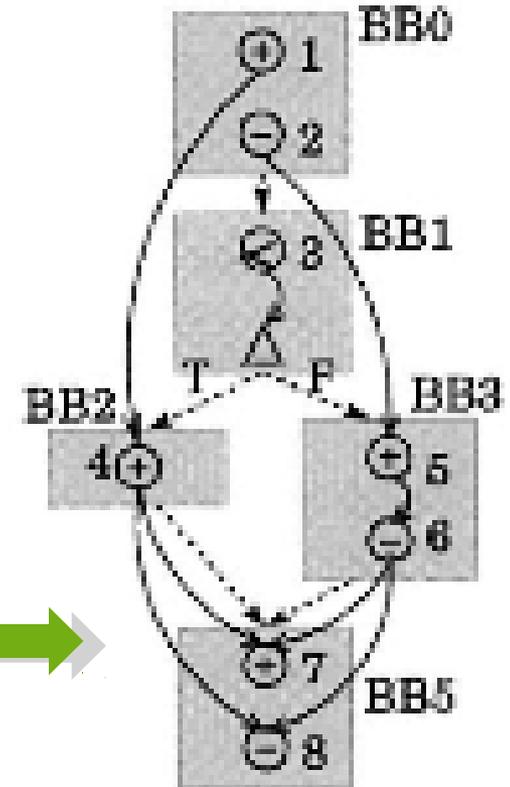


Kontrollfluss-  
Graph (CFG)

Kombiniert:  
CDFG



Datenfluss-  
Graph (DFG)



# CDFGs

---

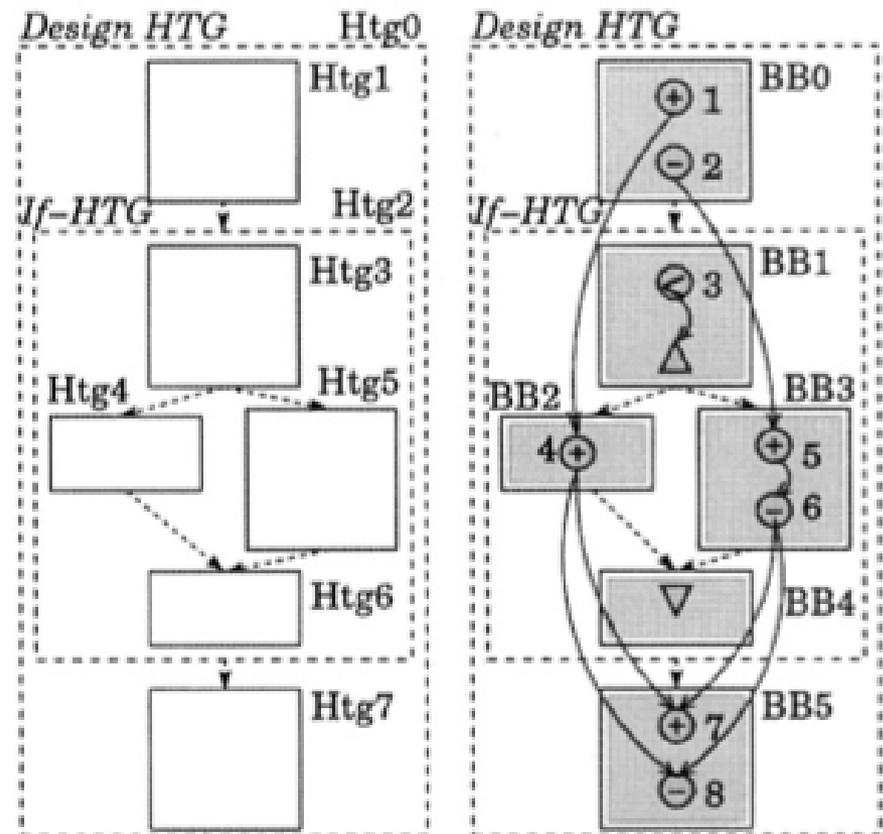
- CDFGs sind ausdrucksstärker als (reine) DFGs
  - Darstellung von Schleifen mit variablen Grenzen (unbounded loops) möglich
- Parallelität nur innerhalb von Basisblöcken explizit
- Zusätzliche Analysen erforderlich, um Parallelität zwischen einzelnen Basisblöcken zu bestimmen
  - Abrollen von Schleifen (*loop unrolling*)
  - Fließbandverarbeitung (*loop pipelining*)
  - Verbinden von Schleifen (*loop merging*)
  - Kacheln von Schleifen (*loop tiling*)
- Ziele: Optimierung von...
  - Latenzen oder Durchsatz
  - Größe und Anzahl der Speicherzugriffe
- Automatisch oder benutzergesteuert realisierbar

# Hierarchische Taskgraphen

- In CFGs lassen sich zu Kontrollflussabhängigkeiten auch Datenabhängigkeiten zwischen Basisblöcken hinzufügen
- Beispiel:  
Hierarchische Taskgraph-Darstellung in SPARK [4]

```

1: t = a + b;
2: u = a - b;
3: if (a < b)
4:   v = t + c;
   else
5:   w = u + c;
6:   v = w - d;
7:   x = v + e;
8:   y = v - e;
    
```



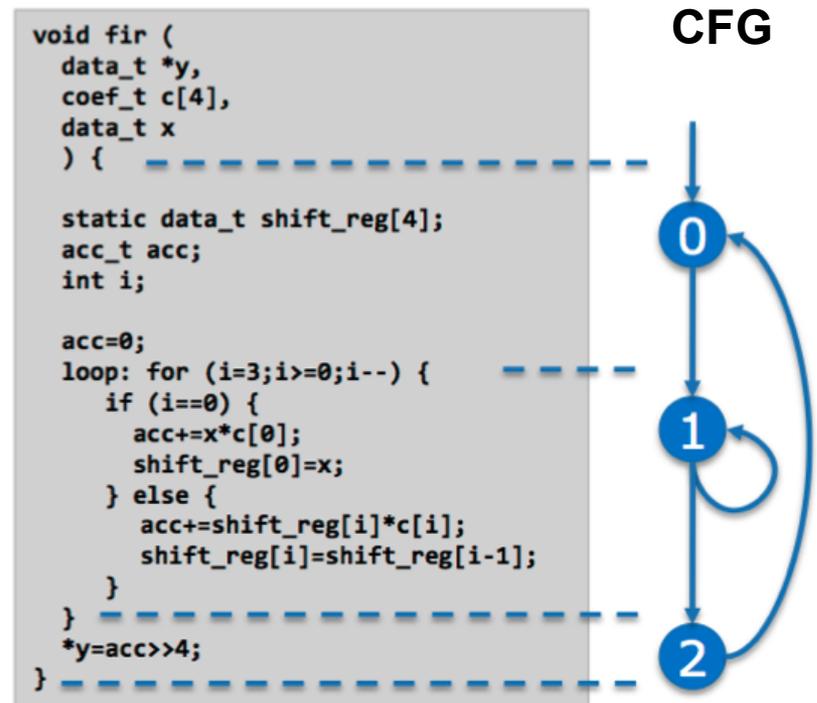
# Kontroll- und Datenflussextraktion

---

- Erster Syntheseschritt: Kontrollflussextraktion
  - Kontrollfluss enthalten in Schleifen und bedingten Sprüngen des Codes
- Eintritt in und Austritt aus einer Schleife entspricht Eintritt in und Austritt aus einem Zustand eines endlichen Automaten (in RTL-Beschreibung)
  - Durch Optimierungen evtl. keine exakte Zuordnung von Schleifen zu Zuständen
- Mehrfache Schleifen führen evtl. zu mehreren Taktzyklen
  - Oft Optimierung in Synthesewerkzeugen: Minimierung der Taktzyklen ohne Codeänderungen

# Kontrollflussextraktion: Beispiel

- Annahme: jede Operation benötigt 1 Taktzyklus/Zustand
- Real: oft mehrere Zyklen benötigt
  - z.B. Expansion von Zustand 1 zu Unterzuständen 11, 12 und 13
  - Steuerlogik wird durch erzeugte Protokolle der Schnittstellensynthese beeinflusst
  - Synthesewerkzeuge erzeugen komplexere oder optimierte Automaten



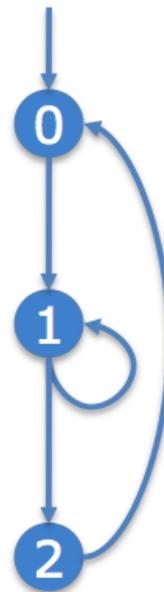
# Datenflussextraktion: Beispiel

- Datenflussextraktion: vergleichsweise einfach
  - Abrollen aller Schleifen
  - Auswertung bedingter Anweisungen

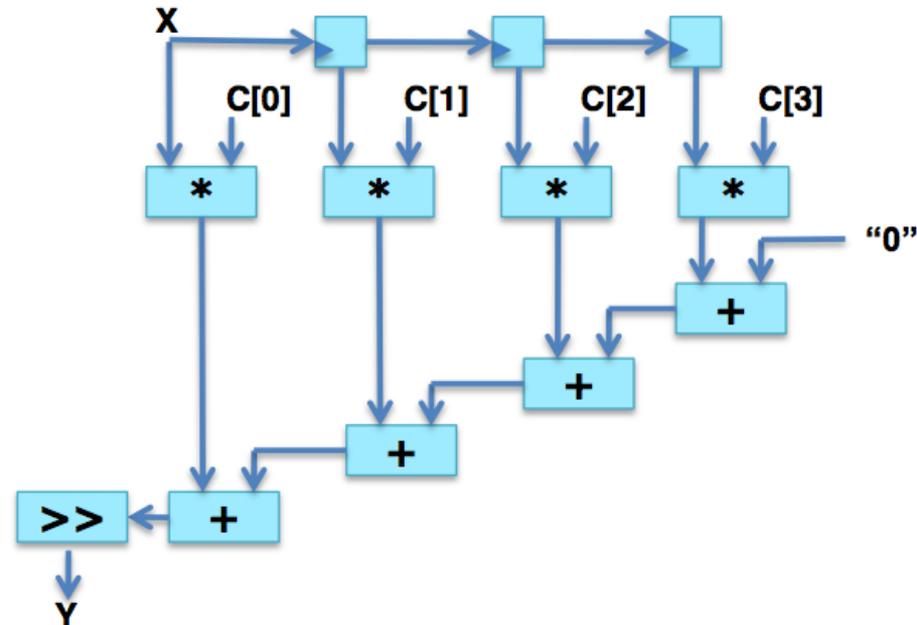
```
void fir (
  data_t *y,
  coef_t c[4],
  data_t x
) {
  static data_t shift_reg[4];
  acc_t acc;
  int i;

  acc=0;
  loop: for (i=3;i>=0;i--) {
    if (i==0) {
      acc+=x*c[0];
      shift_reg[0]=x;
    } else {
      acc+=shift_reg[i]*c[i];
      shift_reg[i]=shift_reg[i-1];
    }
  }
  *y=acc>>4;
}
```

Control Behavior

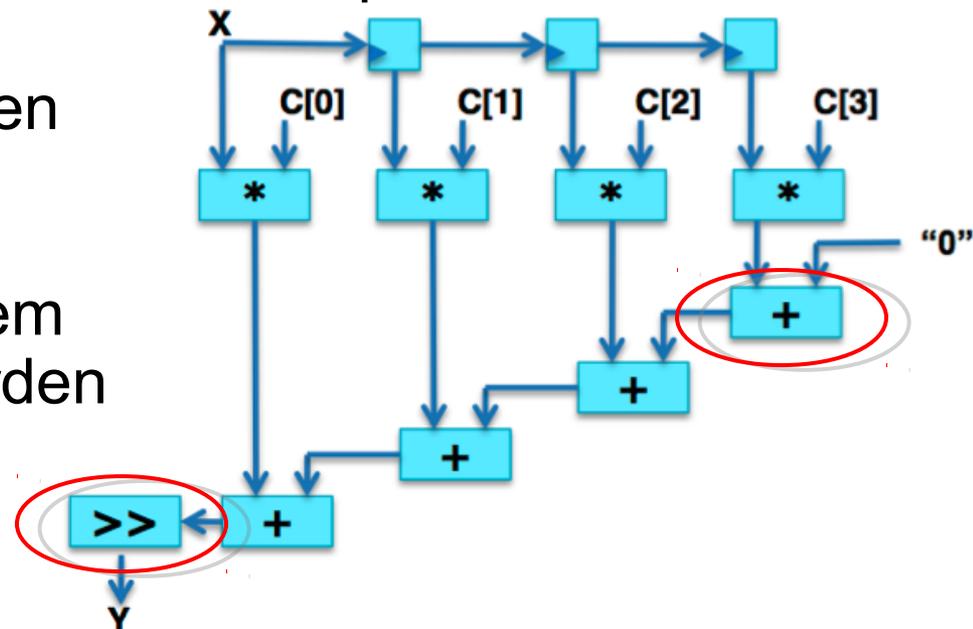


Datapath Behavior



# Datenflussextraktion: Optimierungen

- Endgültiger Datenpfad in RTL-Implementierung komplexer als dargestellt
- Grund: Optimierungen durch Synthesewerkzeug
  - z.B. erster Addierer nicht benötigt
  - Letzte Schiebeoperation ist Zweierpotenz
- Komplexere Optimierungen beim Scheduling
  - Entscheidung, welche Operationen in welchem Zyklus ausgeführt werden



# Allokation

---

- Definition der **Arten** und **Anzahlen** von **HW-Ressourcen**
  - Funktionseinheiten, Speicher, Verbindungskomponenten
- Komponentenauswahl aus RTL-Komponentenbibliothek
  - Wichtig: mind. 1 Komponente für jede Operation im Spezifikationsmodell auswählen
- Bibliothek beinhaltet auch Komponenteneigenschaften
  - Fläche, Verzögerung, Energie
  - Metrik von Komponenten
  - Verwendung durch andere Syntheseschritte

# Allokation

---

- Einige Komponenten evtl. in Folgeschritten hinzugefügt
  - Während Scheduling und Bindung
  - z.B. können Verbindungskomponenten (Busse oder Punkt-zu-Punkt-Verbindungen) vor oder nach Scheduling oder Bindung hinzugefügt werden

# Scheduling

---

- Alle in der Spezifikation vorgegebenen Operationen müssen Zyklen zugeordnet werden
- Für jede Operation wie z.B.  $a = b \text{ op } c$ :
  1. Lesen der Variablen  $b$  und  $c$  aus ihren Quellen
    - Speicherelemente oder Funktionseinheiten
  1. Weiterleiten der Werte an Eingang einer Funktionseinheit, die Operation **op** ausführen kann
  2. Weiterleiten des Ergebnisses an vorgegebenes Ziel
    - Speicherelement oder Funktionseinheit

# Scheduling

---

- Scheduling einer Operation über einen Zyklus oder mehrere Zyklen
  - Abhängig von der Funktionseinheit, auf die abgebildet wird
- Operationen können **verkettet** werden
  - Ausgabe von Operation=direkt Eingabe der folgenden
- Operationen können **parallel** ausgeführt werden
  - Wenn keine Datenabhängigkeiten zwischen ihnen existieren
  - Solange ausreichend viele Ressourcen gleichzeitig zur Verfügung stehen

# Bindung

---

- Jede *Variable*, die Werte über Zyklen hinweg behält, muss an eine *Speichereinheit (SE)* gebunden sein
  - Variablen mit nicht überlappenden/gegenseitig ausschließenden Lebenszeiten können auf die selbse Speichereinheit gebunden werden
- Jede *Operation* im Spezifikationsmodell muss an eine *Funktionseinheit (FE)* gebunden werden, die die Operation ausführen kann
  - Existieren mehrere dieser FEs, muss der Bindungsalgorithmus diese Auswahl optimieren
- Bindung von SE und FE auch von Bindung an Verbindungskomponenten abhängig
  - Jeder Transfer zwischen Komponenten muss an Verbindungskomponente, z.B. Bus oder Multiplexer, gebunden sein

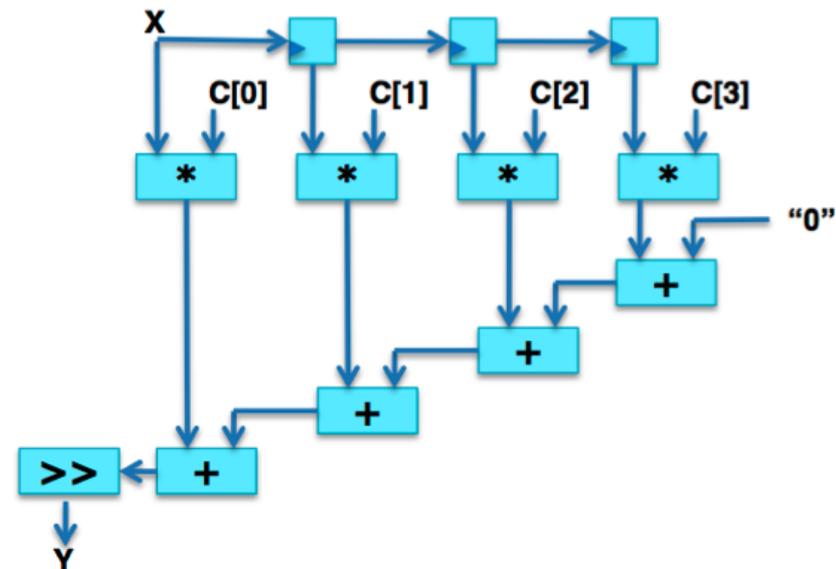
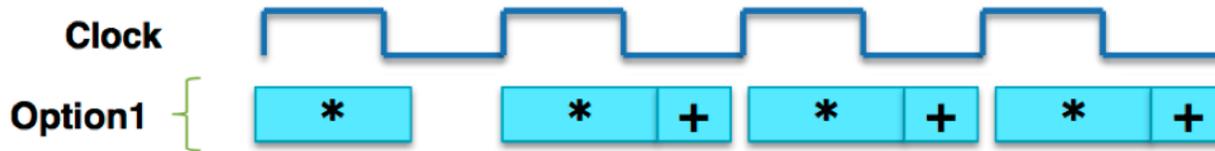
# Scheduling und Bindung

---

- Synthesewerkzeug legt beim Scheduling fest, in welchem Zyklus welche Operationen stattfinden sollen
- Berücksichtigung verschiedener Parameter, z.B.:
  - Taktfrequenz
  - Taktvarianzen
  - Zeitinformationen aus der Technologiebibliothek
  - Latenzen
  - Durchsatzanforderungen
- Für viele Entwürfe sind mehrere verschiedene Implementierungen realisierbar

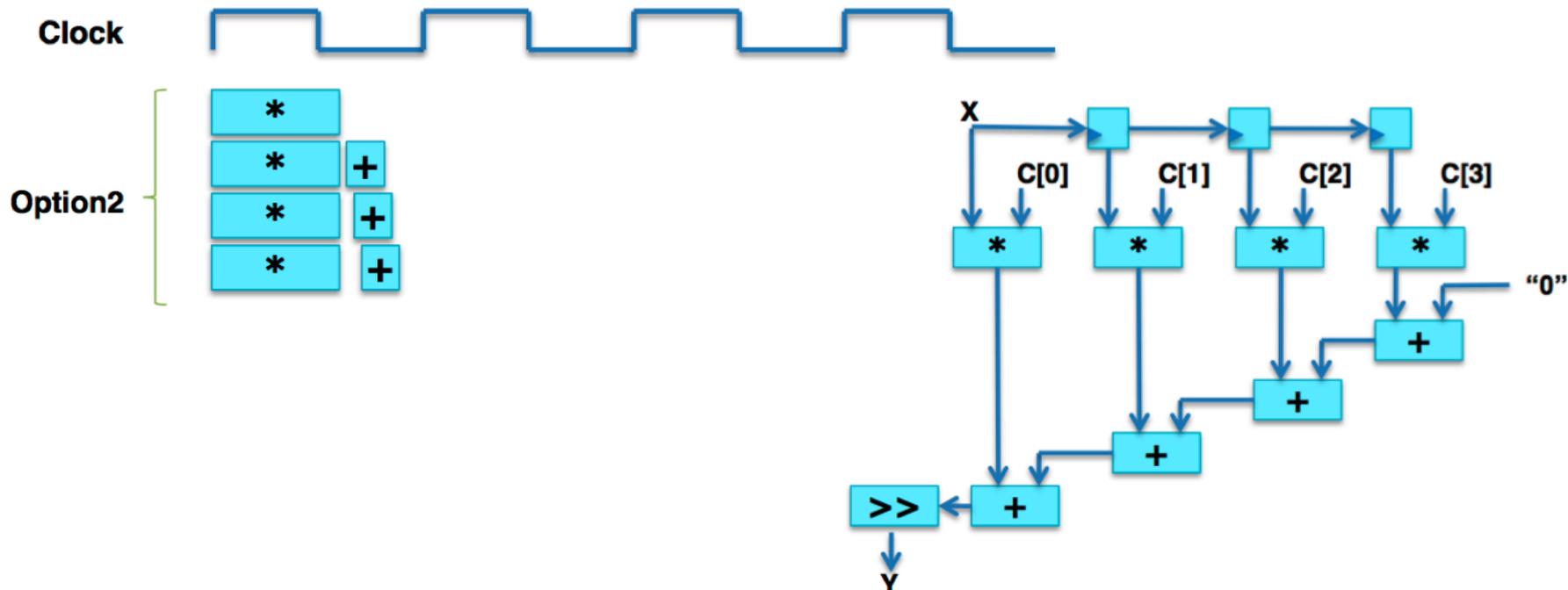
# Beispiel: Scheduling und Bindung

- Implementierungsvariante 1:
  - Je ein Addierer und ein Multiplizierer
  - 4 Taktzyklen benötigt



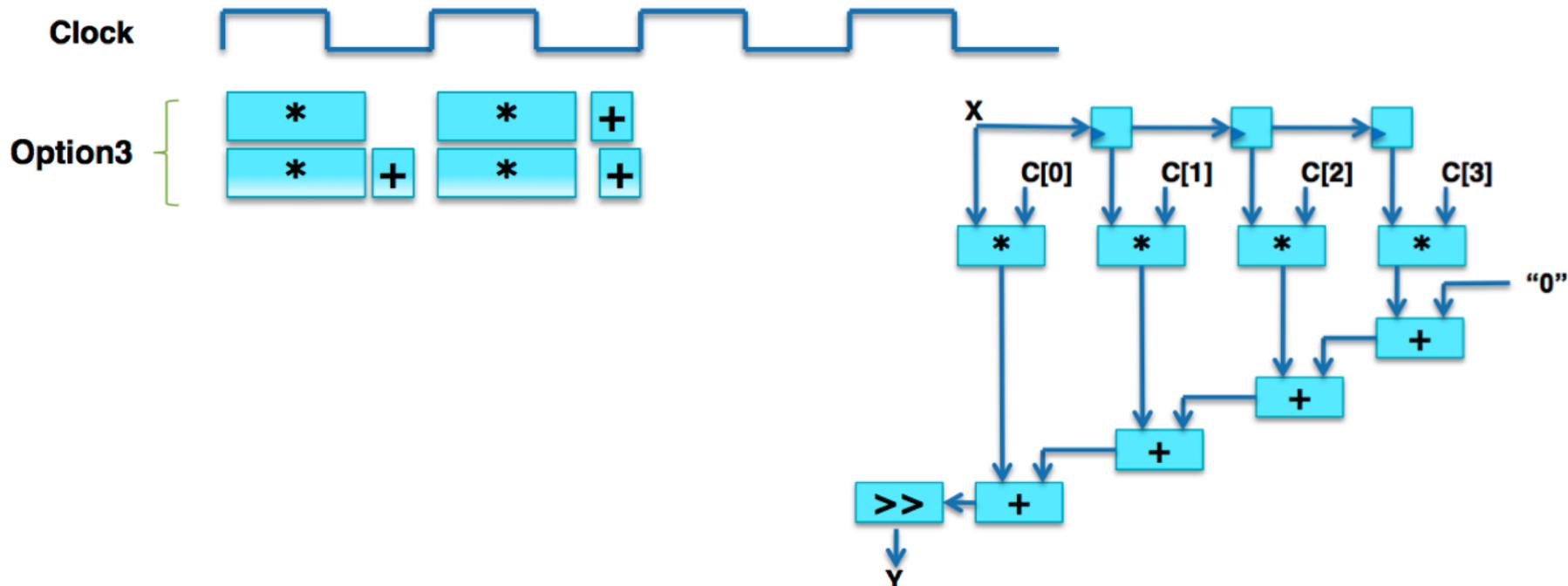
# Beispiel: Scheduling und Bindung

- Implementierungsvariante 2:
  - 3 Addierer und 4 Multiplizierer
  - 1 Taktzyklus
  - Voraussetzung: Addiererkette in 1 Zyklus ausführbar
  - Schneller, aber auch größer



# Beispiel: Scheduling und Bindung

- Implementierungsvariante 3:
  - Zwei Addierer und zwei Multiplizierer
  - Zwei Taktzyklen
  - Kleiner als Variante 2, schneller als Variante 1





# Generierung: Datenpfad

---

- **Datenpfad = Menge von Speicherelementen**
  - Register, Registerbänke, Speicher
- **+ Menge von Funktionseinheiten**
  - ALUs, Multiplizierer, Schieberegister, Spezialfkt.
- **+ Verbindungskomponenten**
  - z.B. Tristate-Treiber, Multiplexer, Busse
- Alle (RTL-)Komponenten in unterschiedlichen Anzahlen und Arten allozierbar
  - Können beliebig durch Busse verbunden werden
- Jede Komponente...
  - benötigt ein oder mehrere Zyklen zur Ausführung
  - kann Fließbandverarbeitung (*Pipelining*) nutzen
  - kann Ein- und Ausgaberegister besitzen
- Auch der gesamte Datenpfad und der Controller können Pipelining mit mehreren Stufen verwenden

# Generierung: Datenpfad

---

- Primäre Ein-/Ausgabeports des Systems verbinden mit der Außenwelt
  - Übertragung von Daten und Steuerinformationen
  - Verwendung für Protokolle und Synchronisation
  - Daten-Ein-/Ausgänge mit Datenpfad verbunden
  - Steuerungs-Ein-/Ausgänge mit Steuerung verbunden
  - Zudem: Steuersignale von Steuerung zu Datenpfad und Statussignale von Datenpfad zur Steuerung
- Einige Architekturen nutzen nur Teilmenge der Verbindungen
  - Einige Steuerfunktionen als Teil des Datenpfads implementierbar
    - z.B. ein Zähler zusammen mit anderer Logik im Datenpfad, die Steuersignale erzeugt

# Generierung: Steuerung

---

- Steuerung (*Controller*): Endlicher Automat
  - Kontrolliert Datenfluss im Datenpfad durch Setzen von Steuersignalen
    - Menge von Steuersignalen: Steuerwort (*control word*)
  - z.B. Auswahlsteuerungs-Eingänge von Funktionseinhalten, Registern und Multiplexern
- Eingangsdaten für die Steuerung
  - Primäre Eingänge des Systems (*control inputs*)
  - Ausgänge von Datenpfadkomponenten, z.B. Vergleichen (*status signals*)

# Generierung: Steuerungskomponenten

---

- Steuerung besteht aus:
  - Statusregister (SR)
  - Logik zur Berechnung des Folgezustands
  - Ausgabelogik
- SR speichert aktuellen Prozessorzustand
  - Entspricht dem aktuellen Zustand des endlichen Automaten, der die Abläufe der Steuerung beschreibt
- Folgezustandslogik berechnet den nächsten in das SR zu ladenden Zustand
- Ausgabelogik erzeugt Steuersignale und Steuerausgaben

# RTL-Modell

- Architekturbeschreibung in RTL in verschiedenen Detaillierungsgraden
  - Entsprechend den Entscheidungen bei der Bindung
  - Komponenten mit oder ohne Bindung angegeben
- Beispiel:  
Möglichkeiten für die Bindung der Operation  $a = b + c$  in Zustand  $n$

Without any binding:

```
state (n) : a = b + c;  
go to state (n + 1);
```

With storage binding:

```
state (n) : RF(1) = RF(3) + RF(4);  
go to state (n + 1);
```

With functional-unit binding:

```
state (n) : a = ALU1 (+, b, c);  
go to state (n + 1);
```

With storage and functional-unit binding:

```
state (n) : RF(1) = ALU1 (+, RF(3), RF(4));  
go to state (n + 1);
```

With storage, functional-unit, and connectivity binding:

```
state (n) : Bus1 = RF(3); Bus2 = RF(4);  
Bus3 = ALU1 (+, Bus1, Bus2);  
RF(1) = Bus3;  
go to state (n + 1);
```

# RTL-Modell

---

- Falls die RTL-Beschreibung nur eine teilweise Bindung von Ressourcen verwendet, muss die nachfolgende Logiksynthese (-> ISE) Bindung und Optimierung durchführen
  - Damit ist bessere Anpassung an Zeitmodelle für die Zielplattform möglich, z.B. durch Berücksichtigung von Leitungskapazitäten durch das spezifische Layout und die entstehenden Laufzeiten

# Entwurfsfluss-Variationen

---

- Allokation, Scheduling und Bindung können gleichzeitig oder in verschiedenen Ausführungsreihenfolgen ablaufen
  - Abhängig von verwendeten Strategien und Algorithmen
  - Allerdings bestehen immer Abhängigkeiten zwischen diesen Phasen
- Bei gleichzeitiger Ausführung wird der Synthesevorgang oft zu komplex für reale Entwürfe

# Entwurfsfluss-Variationen

---

- Ausführungsreihenfolge abhängig von Entwurfsparametern und Optimierungszielen des Synthesewerkzeugs
  - z.B. zuerst Allokation, wenn Scheduling bei begrenzten Ressourcen Latenzen minimieren oder Durchsatz maximieren soll
  - Allokation wird während Scheduling bestimmt, wenn Scheduling die Fläche unter Zeitbeschränkungen minimieren soll

# Entwurfsfluss-Variationen

---

- Ressourcenbeschränkte Ansätze werden verwendet,
  - wenn Entwickler Datenpfadarchitektur vorgeben will
  - wenn Anwendung durch FPGA mit beschränkten Ressourcen beschleunigt werden soll
- Zeitbeschränkte Ansätze werden verwendet,
  - wenn Reduktion der Fläche der Schaltung Optimierungsziel ist und ein geg. Durchsatz realisiert werden muss
    - z.B. in Multimedia- oder Telekommunikationsanwendungen
- Ressourcenbeschränkte Probleme können durch zeitbeschränkte Ansätze gelöst werden und umgekehrt
  - Aufweichung der Zeitbedingungen, bis die benötigte Fläche klein genug ist

# Entwurfsfluss-Variationen: Ziele

---

- Wichtige klassische Entwurfsziele:
  - Latenz, Durchsatz, Ressourcenverbrauch und Fläche
  
- Neue Entwurfsziele:
  - Taktperiode
  - Speicherbandbreite
  - Speicherabbildung
  - Energieverbrauch
  - ...USW.
  
- Besonders wichtig für eingebettete Systeme
- Neue Ziele erschweren die Entwurfsaufgabe weiter

# Zusammenfassung

---

- Überblick: Synthese
- High-Level-Synthese
- Werkzeugfluss
- Übersetzung und Modellierung
  - Kontroll-/Datenfluss
- Allokation
- Scheduling
- Bindung
- Generierung
- Ausgabemodell

# Literatur

---

- a. P. Coussy, D. Gajski, M. Meredith, A. Takach:  
An Introduction to High-Level Synthesis  
IEEE Design & Test of Computers
- b. P. Coussy, A. Morawiec (Editors):  
High-Level Synthesis – from Algorithm to Digital Circuit  
Springer 2008
- c. G. DeMicheli:  
Synthesis and Optimization of Digital Circuits  
McGraw-Hill Higher Education 1994  
ISBN:0070163332
- d. Gupta:  
SPARK: A Parallelizing Approach to the High-Level Synthesis of Digital  
Circuits  
Kluwer 2004