1.3 Implementierung eingebetteter Systeme

Zur Vorlesung

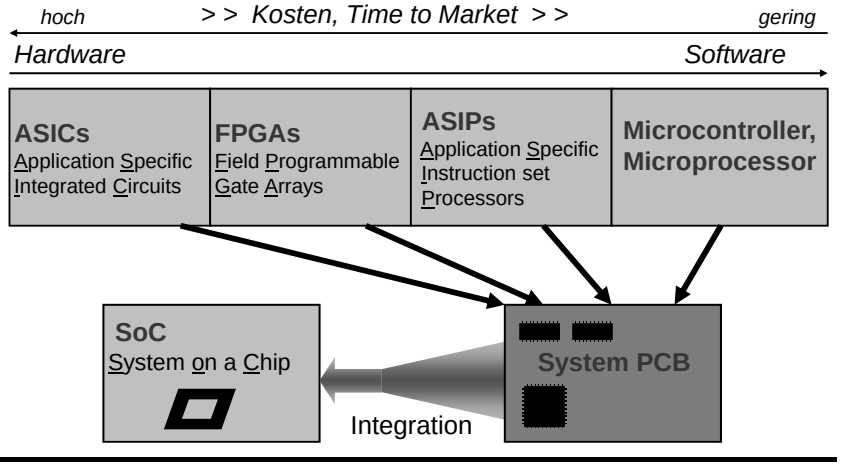
Embedded Systems

WS 14/15

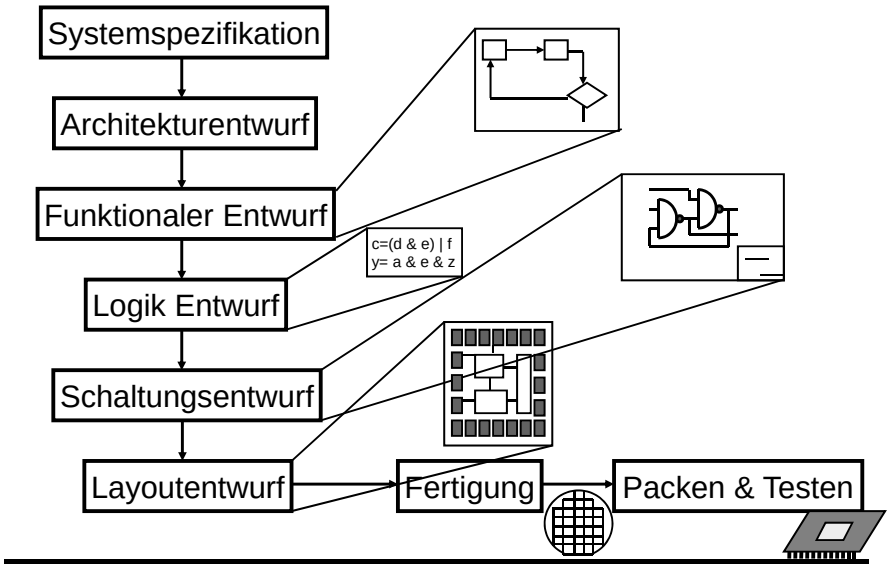
Reiner Kolla

1.3.1 Implementierungsplattformen im Überblick

Zur Implementierung eingebetteter Systeme gibt es eine Fülle von Möglichkeiten und Kombinationen von Hardware und Software.

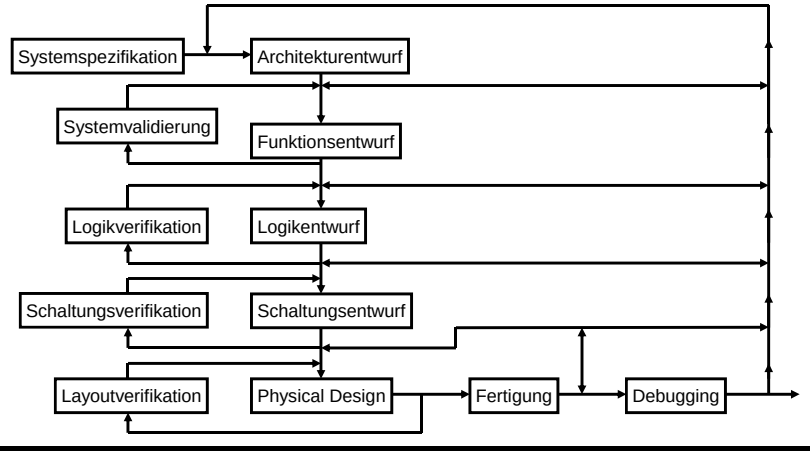


1.3.2 ASIC Design Zyklus



ASIC Designzyklus ff

Alle Phasen des Entwurfs sind durch Werkzeuge unterstützt, z.T. sogar voll automatisiert.
Synthesewerkzeuge sorgen für den Übergang zwischen den Ebenen,
Analysewerkzeuge validieren Übergänge.

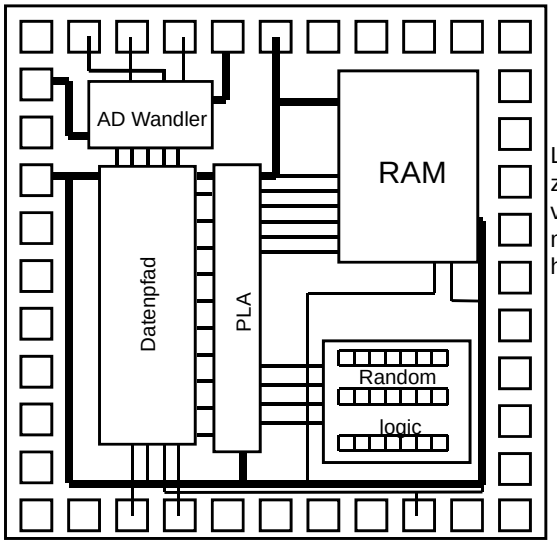


ASIC Entwurfsstile

Um das Problem der automatischen Generierung von Fertigungsdaten für ASICs zu vereinfachen, hat man verschiedene Entwurfsstile entwickelt, mit folgenden Merkmalen

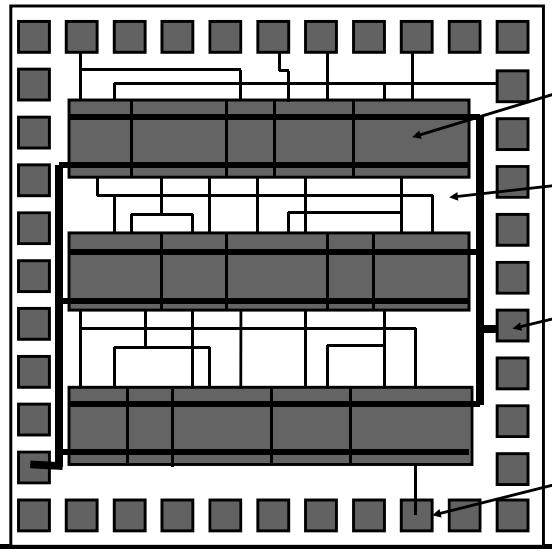
- Vorgaben von Fertigungsmasken
- Voranfertigung bis auf Metallagen
- Komplette Voranfertigung, Layout durch Programmierung
- Benutzung von Generatorprogrammen für RAM,PLA…
- Benutzung von großen, konfigurierbaren Subsystemen (Cores)

Full Custom Entwurf



Layout aller Masken,
z.T. generierte Blöcke,
vorgefertigtes Layout,
mit speziellen Tools
hergestelltes Layout,

Standardzellen



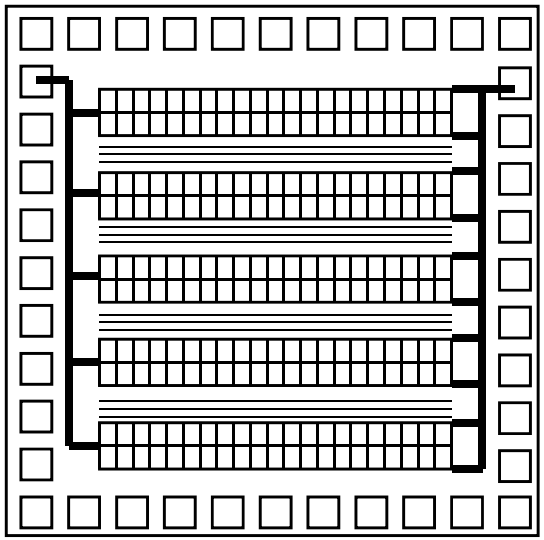
Standardzelle
(Layout vom Hersteller)

Verdrahtungs
kanal
(vom Layoutwerkzeug)

Versorgungs
pad

Pad

Gatearray

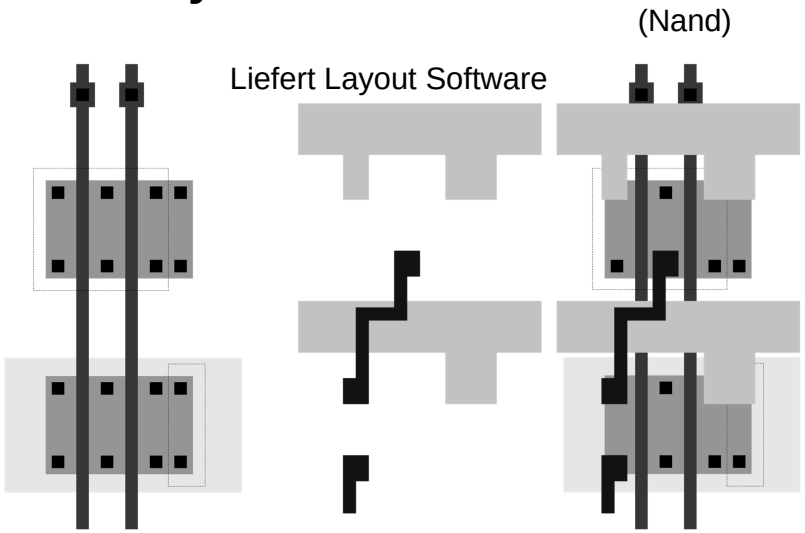


Transistorraster
vorgefertigt

Kanalraster
z.T. vorgefertigt

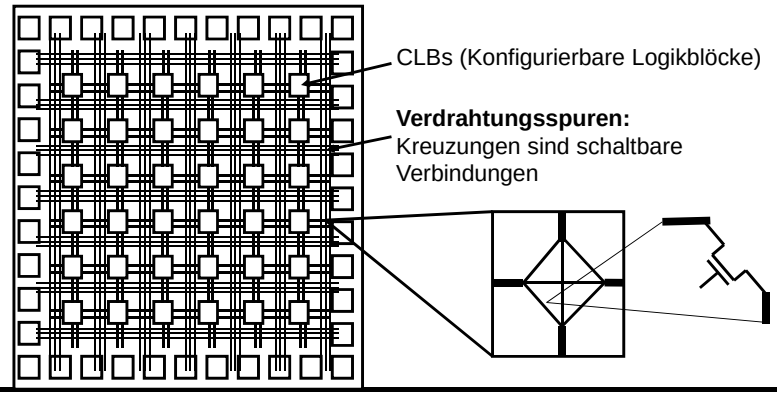
Funktion wird
allein durch
die Metallschichten
bestimmt
MPGA
mask programmable gate
array

Gatearrayzelle



FPGAs

Fertigstellung des Chips erfolgt nur noch durch elektrische Programmierung von Schaltern, entweder irreversibel durch sog. Antifuse Verbindungen oder rekonfigurierbar durch SRAM-Zelle + Schalttransistor.



1.3.3 ASIPs

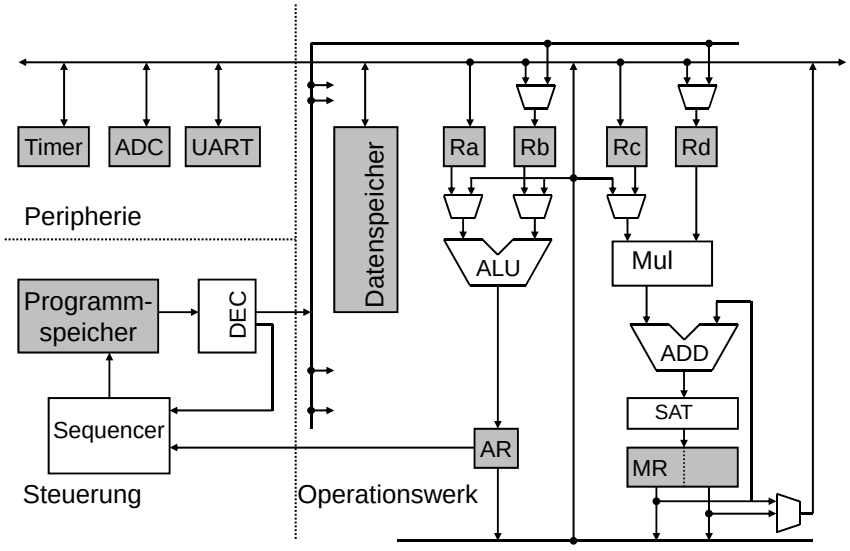
Bei den bisher vorgestellten Techniken ist der Entwurf hardware-orientiert, wenngleich bei FPGAs die Fertigung durch reine Programmierung erfolgt.

Die Nahtstelle zwischen hardware- und softwareorientiertem Entwurf bilden **ASIPs** = application specific instruction set processors.

- **hohe Konfigurierbarkeit**, parametrisiert in
  - Registerzahl
  - Programm und Datenspeichergröße
  - Wortbreite
  - Operationen des Datenpfades
  - Zahl und Art von Peripheriekomponenten
- **Einsatzgebiet**

  Einsatz erfolgt, wenn „Off the Shelf Prozessoren“ zu groß sind, zu viel Leistungsaufnahme haben, oder der speziellen Anwendung in der Performanz nicht gewachsen sind.

ASIPs: Beispielhafte Skizze



ASIPS -- ff

Programmspeicher enthält das Programm des Prozessors über

- Mikroinstruktionen
  - Kodieren direkt alle Steuerungsmöglichkeiten der Datenpfade,
    - **horizontal,** pro Leitung ein Bit, hohe Parallelität
    - **vertikal**, kompakter Code, nur wenige Kombinationen
- Makroinstruktionen
  - Umsetzung der Instruktionen über mehrere Zyklen (zusätzliche Steuerungslogik nötig)
- Wortbreite ergibt sich aus
  - Instruktionskodierung
  - Zahl der Operatoren und Register im Operationswerk
  - Zahl und Art der Peripheriekomponenten

Der Instruktionssatz eines solchen Prozessors ist also nicht fest, sondern von der aktuell eingestellten Konfiguration abhängig.

➔ Problem

Hochsprachen, Compiler kaum verfügbar.

Programmierung in Maschinen- bzw. Microcode.

1.3.4 Microcontroller

Microcontroller sind Mikroprozessoren, die speziell im Hinblick auf den Einsatz in eingebetteten Systemen optimiert sind. Wir wollen die wichtigsten Merkmale gegenüber Standardprozessoren herausstellen:

- **Abstriche gegenüber Standardprozessoren**

Aus Kostengründen ist die Komplexität sehr viel niedriger als bei Standardprozessoren. Abstriche werden bei Dingen gemacht, die man in vielen eingebetteten Systemen nicht braucht:

- niedrigere Wortbreiten (8,16, 32 bit)
- keine Floatingpoint Einheiten, FP Operationen softwareemuliert
- keine Speicherhierarchie (Caches …)
- keine Superskalar Architekturen

Microcontroller -- Merkmale

- **Erweiterungen gegenüber Standardprozessoren**

Andererseits tragen Microcontroller durch Erweiterungen vielen Aspekten eingebetter Systeme Rechnung, die von Standard-prozessoren kaum unterstützt werden:

- komplexes Unterbrechungssystem
  - viele Unterbrechungseingänge
  - viele Unterbrechungsebenen, die aktiviert, deaktiviert werden können
  - kurze Reaktionszeiten auf Unterbrechungen
- komplexes integriertes Peripheriesystem
  - programmierbare Timer
  - programmierbare Schnittstellen
  - programmierbare AD/DA Wandler mit entsprechenden Ports
  - Konfiguration der Peripherie durch sog. Special Function Register
- integrierte Programm- und Datenspeicher
- konfigurierbarer Bus (über SFR) zu externen Erweiterungen

Microcontroller -- Architekturmerkmale

- CISC Befehlssatz
  - hohe Codedichte erforderlich (begrenzter Programmspeicher)
  - viele implizite Operanden (Push, Pop, Call, Return Befehle)
  - meist 2 Adresscode
  - Trend zu RISC Befehlssätzen bei 32 bit MCs
- Harvard Architektur
  Motiv: Programmspeicher (Flash ROM, EPROM) und Datenspeicher (SRAM) sind integriert und von ihrer Natur her getrennt. Die Menge der laufenden Programme wird nicht im Betrieb geändert.

Vorteil: Microcontroller sind weitverbreitet, preisgünstig und es existieren viele Entwicklungswerkzeuge dazu (C-Compiler, Assembler, Debugger …)

Nachteil: Beschränkte Performanz

1.3.5 Echtzeitbetriebssysteme

Vor allem im Zusammenhang mit Microcontrollern kann es je nach Komplexität und Aufgabenstellung nützlich sein, auf Programme zurückzugreifen, die die Implementierung auf folgende Weisen unterstützen

- **Verwaltung von einzelnen Rechenprozessen**
  - ⊙ Ablaufplanung
  - ⊙ Zeitmanagement
  - ⊙ Synchronisiation von Prozessen
- **Überwachung und Zuteilung von Ressourcen**
  - ⊙ Peripheriekomponenten
  - ⊙ Speicher

Je nach Komplexität des einzubettenden Systems können diese Aufgaben direkt von einem dazu entwickelten Steuerprogramm (Scheduler) übernommen werden, oder man greift auf ein in seiner Austattung oft konfigurierbares Echtzeitbetriebssystem zurück.

Echtzeitbetriebssysteme -- Systemkern

Man benötigt meist nicht die Vielzahl von Diensten, die gewöhnliche Betriebssysteme bieten (Privilegien, Speicherschutzmechanismen, Dateiverwaltung, Netzwerk), stellt aber an die Verarbeitung von Unterbrechungen und das Zeitmanagement erhöhte Anforderungen.

Selbst Systemfunktionen sollten unterbrechbar sein, um auf Unterbrechungen schnellstmöglich reagieren zu können.

Zur Prozessverwaltung nutzt man daher sogenannte Mikrokerne:

Ein **Mikrokern** ist ein Programm, das aus wenigen, nichtunterbrechbaren Teilen zur Implementierung von Zustandsübergängen auf der Menge der Prozesse besteht, und die zugehörigen Datenstrukturen zur Prozessverwaltung verwaltet.

Prozesse, **Tasks** bzw. **Threads** sind die eigentlichen, z.T. vom Benutzer programmierten, z.T. in Ergänzung zum Kern verfügbaren, Teile des eingebetteten Echtzeitsystems.

Prozesse

Prozesse sind Programme, die als Rechenlast vom Systemkern verwaltet werden.

Klassifikation:

Leichtgewichtig vs. Schwergewichtig:

Leichtgewichtige Prozesse arbeiten gegeneinander ungeschützt auf dem gleichen Datenspeicher. Ihr aktueller Zustand kann durch einen sog. Kontext wohldefiniert werden.

Kontext: Inhalt aller Register (GP und SP Register (PC…))

Schwergewichtige Prozesse unterliegen dagegen einem Schutzkonzept, d.h. zum Kontext gehören zusätzlich Dinge wie zugeteilte Speicherseiten, Privilegien …

Präemptiv vs. Ununterbrechbar:

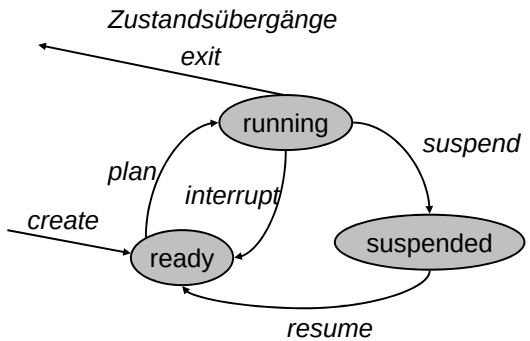
Prozesse, die in ihrer Ausführung unterbrochen werden können, heißen auch präemptive Prozesse.

Im Rahmen eingebetteter Systeme betrachtet man in der Regel nur **leichtgewichtige, präemptive Prozesse** (auch Threads genannt).

Prozesse -- ff

Prozesse werden in folgenden Zuständen verwaltet.

Prozesskontrollblock:



Der Systemkern realisiert die entsprechenden Zustandsübergänge. Zu den Zuständen “ready” und “suspended” verwaltet der Kern eine oder mehrere Warteschlangen, je nach bereitgestellten Mechanismen zu Ablaufplanung und Synchronisation.

Prozessverwaltung

Zu den Aktionen:

create: Erzeugt einen Prozess und reiht ihn in die entsprechende ready queue ein. Häufig hat man eine feste Prioritätshierarchie (0-32) und zu jedem Prioritätswert eine queue, oder eine priority queue.

exit: Systemaufruf zum Beenden. Der Prozess wird aus der Prozessverwaltung entfernt.

interrupt: Unterbrechung des Prozesses

asynchron: von außen eingehende, nicht maskierte Unterbrechung

synchron: vom Prozess per Befehl ausgelöst (Trap)

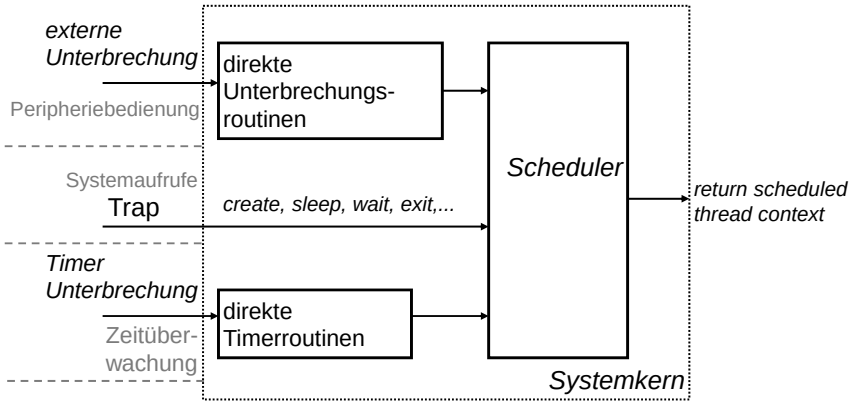
suspend: Ein laufender Prozess wird wegen Unterbrechung suspendiert, oder suspendiert sich per Systemaufruf selbst (Warten auf Timer, Semaphoraufruf, …). Er wird in die Warteschlange eingereiht, die alle suspendierten Prozesse einer festen Suspendierungsursache hält.

resume: Bei Eintreten eines Ereignisses, das eine Suspendierungsursache aufhebt oder ändert (z.B. Timerevent, Semaphoränderung) werden suspendierte Prozesse geprüft und ggf. wieder einer ready queue zugeführt.

Mikrokern -- Grobstruktur

Die meisten Details der Kernfunktionen werden durch Mechanismen definiert, die wir im Verlauf der Vorlesung noch im Zusammenhang mit dynamischer Ablaufplanung kennenlernen.

Grobes Bild der Struktur :



1.4 Fundamentale Probleme bei der Konstruktion eingebetteter Systeme

Zur Vorlesung

Embedded Systems

WS 14/15

Reiner Kolla

Allgemeines Synthesemodell

Unabhängig von der benutzten Implementierungstechnik können wir zunächst folgendes allgemeine Modell eines ES betrachten:

Gegeben ist

- eine Menge von Aufgaben (Operationen, Tasks,…) die untereinander Datenabhängigkeiten besitzen können.
- eine Menge von Ressourcen (ALUs, CPUs, Operatoren,…) auf denen jeweils gewisse Aufgaben ablaufen können.

Gesucht ist

- eine Festlegung der Anzahl von Ressourcen ⇦ **Allokation**
- eine Festlegung des zeitlichen Ablaufs der Bearbeitung der Aufgaben unter Berücksichtigung der Datenabhängigkeiten ⇦ **Ablaufplan (Schedule)**
- eine Zuordnung der Aufgaben zu den Ressourcen, so dass zu jeder Zeit jede Ressource höchstens eine Aufgabe bearbeitet. ⇦ **Bindung (Binding)**

Allgemeines Synthesemodell -- ff

Die Lösung dieser Aufgaben erfolgt je nach Implementierungstechnik sehr unterschiedlich:

Allokation:

Dieses Problem wird in der Regel beim Entwurf gelöst und hat entscheidenden Einfluss auf die Kosten.

Entwurfsraumexploration: untersuche den Einfluss unterschiedlicher Allokationsmöglichkeiten auf die Performanz des Systems

Ablaufplan:

Ablaufpläne können beim Entwurf gelöst werden (statische Planung) oder zur Laufzeit (dynamische Planung), wobei dynamische Lösungen meist bei Softwareimplementierungen, statische eher bei Hardwareimplementierungen genutzt werden.

Bindung:

Bindungen können ebenso als statische, wie dynamische Aufgabe auftreten, und von trivial (eine CPU) bis sehr schwierig (mehrere Ressourcen gleichen Typs mit unterschiedlicher Leistung) rangieren.

Formalisierung der Syntheseprobleme

Definitionen

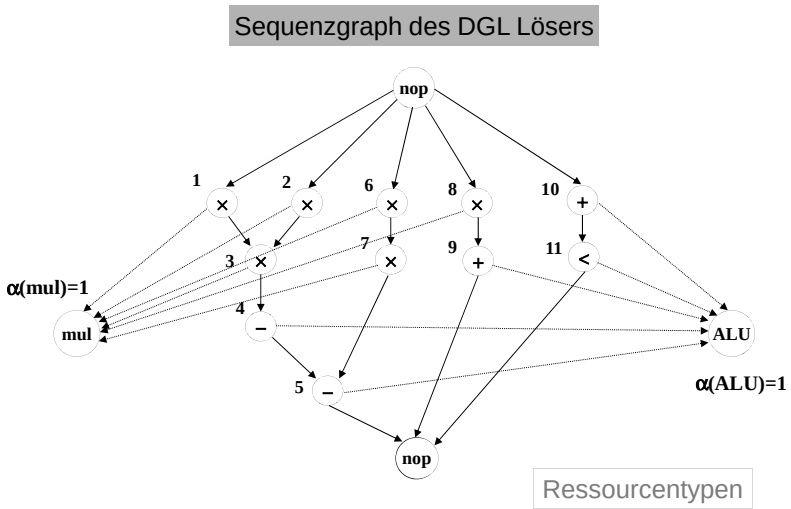
- **Sequenzgraph (Problemgraph, Taskgraph)** $G_S=(V,E)$. Sei ein gerichteter azyklischer Graph mit genau einem Start- und genau einem Endknoten. Im folgenden bezeichnen wir mit $V_S \subseteq V$ die Knotenmenge V ohne den Start- und Endknoten, und mit $E_S \subseteq E$ die Kanten, die weder den Start- noch den Endknoten als Randknoten besitzen.
- Bipartiter **Ressourcengraph** $G_R=(V_R,E_R)$ mit $V_R=V_S \cup V_T$ und $E_R \subseteq V_S \times V_T$. V_T ist die Menge der **Ressourcetypen** (z.B. Prozessor, ALU, Addierer)
- **Kostenfunktion** c: $V_T \rightarrow \boldsymbol{N}_0$, die die Kosten einer Instanz eines Ressourcetypes angibt.
- **Ausführungszeiten** w: $E_R \rightarrow \boldsymbol{N}_0$, die jeder Kante $(v_s,v_t) \in E_R$ die Ausführungszeit der Aufgabe $v_s \in V_S$ auf einer Instanz des Ressourcentyps $v_t \in V_T$ angibt.
- Ein **Syntheseproblem** P ist gegeben durch ein Tupel $P = (G_S,G_R)$

Allokation

Definition -- Allokation

Gegeben sei ein Syntheseproblem. Eine **Allokation** ist eine Funktion α: $V_T \rightarrow \boldsymbol{N}_0$, die jedem Ressourcentyp $v_t \in V_T$ eine Anzahl $\alpha(v_t)$ verfügbarer Instanzen zuordnet.

Sequenz- und Ressourcengraph mit Allokation



Ablaufplan

Nimmt man an, dass jeder Aufgabe jeweils genau ein Ressourcetyp zugeordnet ist, so kann ein Ablaufplan wie folgt vereinfacht definiert werden:

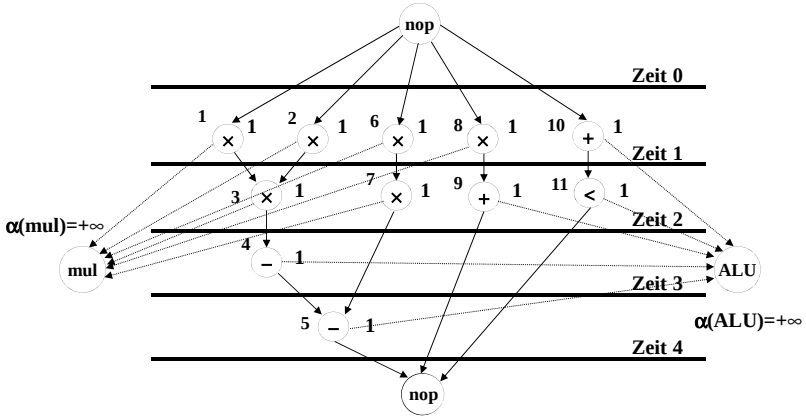
Definition

Gegeben sei ein Syntheseproblem. Ein **Ablaufplan (Schedule)** eines Sequenzgraphen $G_S=(V,E)$ ist eine Funktion $\tau: V_S \rightarrow \mathbf{N}_0$, die jedem Knoten $v_s \in V_S$ die Startzeit $\tau(v_S)$ zuordnet, und für jede Kante $(v_s,v_t) \in E_S$ die Bedingung

$$\tau(v_t) - \tau(v_s) \geq w(v_s)$$

erfüllt. Hierbei gibt $w(v_s)$ die Laufzeit der Aufgabe v_s auf ihrer Ressource an. (Streng genommen wäre es $w(v,\beta(v))$, wobei $\beta(v)$ der Ressourcetyp ist, an den v gebunden wird. Wenn es aber nur einen Ressourcetyp pro Task gibt, ist es einfacher $w(v)$ dafür zu schreiben.)

Ablaufplan: Beispiel



Ausführungslaufzeiten

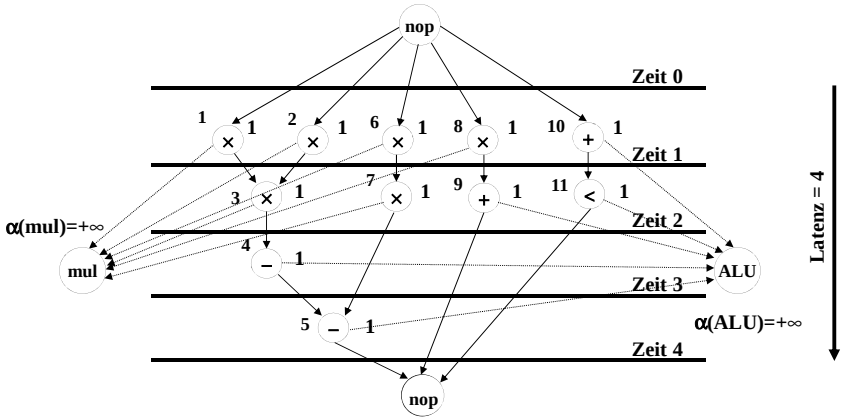
Latenz

Definition

Die Latenz eines Ablaufplans τ eines Sequenzgraphens $G_S=(V,E)$ ist definiert als

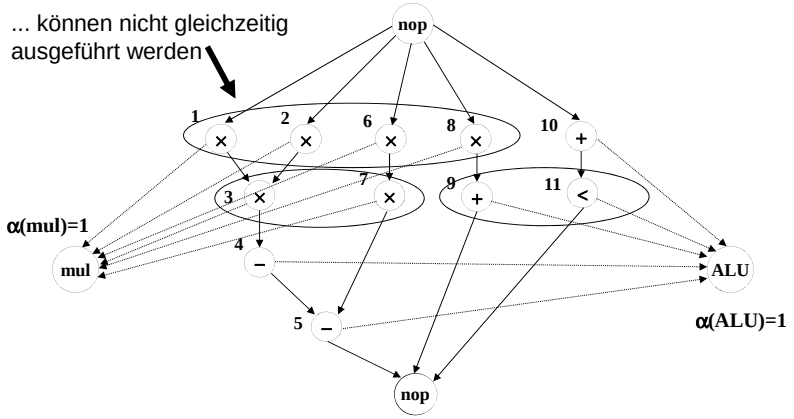
$$L(\tau) = \underbrace{max\{\tau(v_i) + w(v_i); v_i \in V_S\}}_{\text{späteste Beendungszeit}} - \underbrace{min\{\tau(v_j); v_j \in V_S\}}_{\text{früheste Startzeit}}$$

Ablaufplan: Beispiel ff

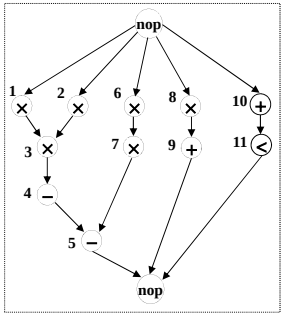


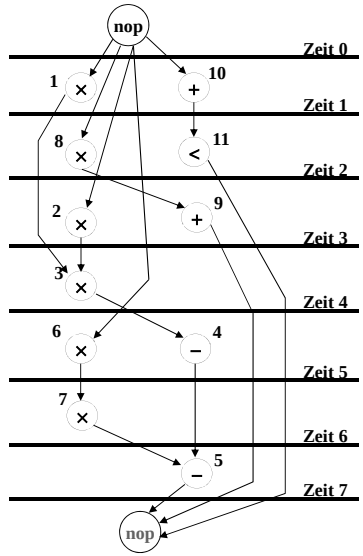
Ausführungslaufzeiten

Ablaufplan bei beschränkter Allokation



Ablaufplan bei beschränkter Anzahl von Ressourcen





Bindung

Definition

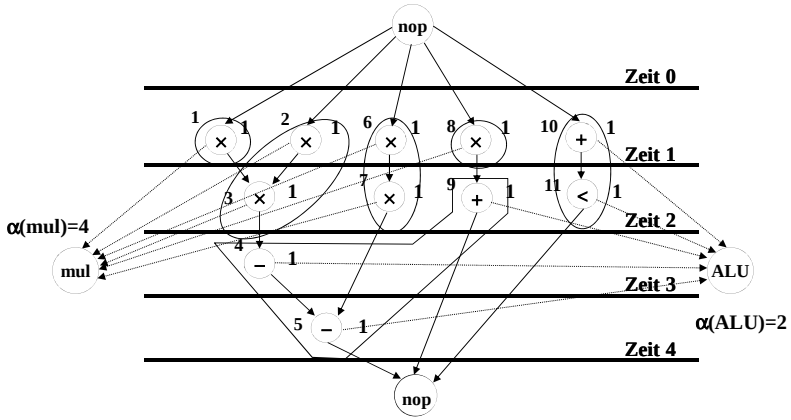
Eine **Bindung** eines Sequenzgraphens $G_S=(V,E)$ bzgl. eines Ressourcengraphen $G_R=(V_R,E_R)$ und einer Allokation $\alpha: V_T \rightarrow \boldsymbol{N}_0$ ist ein Paar von Funktionen $\beta: V_S \rightarrow V_T$ und $\gamma: V_S \rightarrow \boldsymbol{N}$ mit

- $\forall v_s \in V_S$: $(v_s, \beta(v_s)) \in E_R$
- $\forall v_s \in V_S$: $\gamma(v_s) \leq \alpha(\beta(v_s))$

d.h. die Bindung ordnet jeder Aufgabe eine verfügbare Instanz einer Ressource zu, die diese Aufgabe ausführen kann.

Liegt ein Ablaufplan vor, so muss ferner gelten, dass zu jedem Zeitpunkt *t* jeder Ressource nur eine Aufgabe zugeordnet ist.

Bindung: Beispiel



Ausführungslaufzeiten