4.2 Hardwaresynthese

Zur Vorlesung

Embedded Systems

WS 14/15

Reiner Kolla

4.2.1 Direkte Ableitung einer Implementierung

Wir wollen zunächst zeigen, dass wir aus einem gegebenen Problemgraphen mit gültigem Ablaufplan, Allokation und Bindung eigentlich direkt eine Hardwareimplementierung ableiten können:

Gegeben sei

- **Sequenzgraph (Problemgraph, Taskgraph)** $G_S=(V_S,E_S)$.
- **Ressourcengraph** $G_R=(V_R,E_R)$ mit $V_R=V_S \cup V_T$ und $E_R \subseteq V_S \times V_T$.
- **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 auf einer Instanz des Ressourcentyps $v_t \in V_T$ angibt.
- **Allokation** α: $V_T \rightarrow \boldsymbol{N}_0$, die jedem Ressourcentyp $v_t \in V_T$ eine Anzahl $\alpha(v_t)$ verfügbarer Instanzen zuordnet.
- **Bindung** $\beta: V_S \rightarrow V_T$ und $\gamma: V_S \rightarrow \boldsymbol{N}$ bezüglich der Allokation α d.h.: $\forall v_s \in V_S$: $(v_s,\beta(v_s)) \in E_R$ sowie $\forall v_s \in V_S$: $\gamma(v_s) \leq \alpha(\beta(v_s))$
- **Ablaufplan (Schedule)** $\tau: V_S \rightarrow \boldsymbol{N}_0$, der gültig ist unter der Allokation α und Bindung (β, γ)
- **Registerbindung** $\rho: V_S \rightarrow R$ an eine Registermenge R, gültig für den Ablaufplan τ

Multiplexerbasierte Implementierung

Die flexibelste, aber auch teuerste Implementierung basiert auf einer systematischen Verbindung der Register und der alloziierten Ressourcen mittels Multiplexern:

- Betrachte zu jeder Kante $(u,v) \in E_S$ eine Operandenzuordnung $1 \leq op(u,v) \leq indeg_S(v)$, die angibt, zu welchem Operanden die Abhängigkeit korrespondiert.
- Betrachte zu jeder alloziierten Ressource (w,i), $1 \leq i \leq \alpha(w)$, $w \in V_R$ und jedem Operanden j dieser Ressource die Teilmenge von Registern

$$OP(w,i,j) := \{ r \mid \exists (u,v) \in E_S \text{ mit } \rho(u) = r, (\beta(v),\gamma(v))=(w,i) \text{ und } op(u,v) = j \}$$

Wir nehmen an, dass die Register in $OP(w,i,j)$ jeweils mit aufsteigender Nummer geordnet sind, d.h.

$$OP(w,i,j) := \{ r_{i0}, \ldots, r_{ik-1} \}, i_0 \leq \ldots \leq i_{k-1}$$

und setzen $Opindex[w,i,j,r] := s$, falls $r = r_{is}$

Multiplexerbasierte Implementierung -- ff

Eine ähnliche Indizierung nehmen wir bei den Ressourceninstanzen (w,i) vor:

- Betrachte zu jedem Register die Teilmenge von Ressourcen

$$RES(r) := \{ (w,i) \mid \exists u \in V_S \text{ mit } \rho(u) = r, (\beta(u),\gamma(u))=(w,i) \}$$

Wir nehmen an, dass die Ressourceninstanzen ebenfalls durchnummeriert und in $RES(r)$ aufsteigend geordnet sind, d.h.

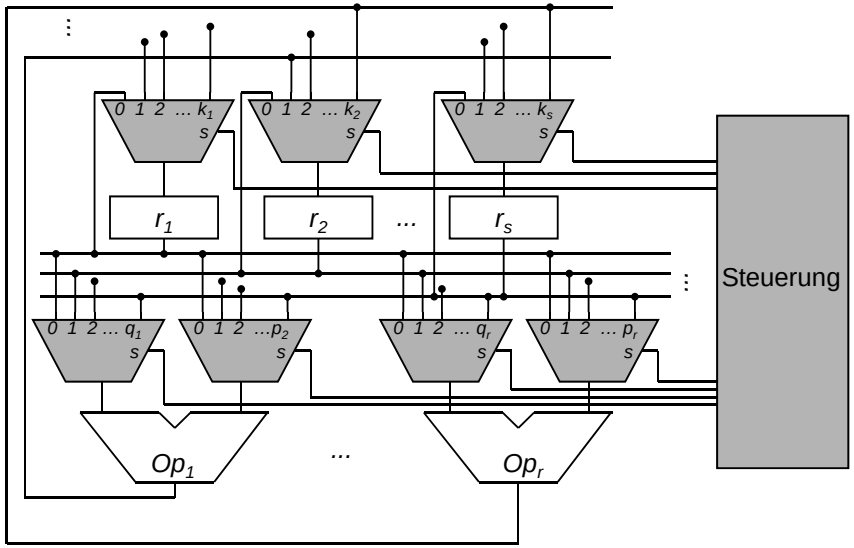
$$RES(r) := \{ (w_1,i_1), \ldots, (w_k,i_k) \}, (w_1,i_1) \leq \ldots \leq (w_k,i_k)$$

und setzen $Rindex[r,w,i] := s$, falls $(w,i) = (w_s,i_s)$.

Wir kommen damit direkt zu folgender Architektur:

- Jedem Register wird ein $\#RES(r)+1$ zu 1 - Multiplexer vorgeschaltet, der über eine Steuerwert selin(r) die entsprechende Ressourceninstanz zum Register weiterleitet.
- Jeder Ressourceninstanz (w,i) wird für jeden Operand j ein $\#OP(w,i,j)$ zu 1 - Multiplexer vorgeschaltet, der das Operandenregister über einen Steuerwert selop(w,i,j) durchschaltet.

Multiplexerbasierte Implementierung -- ff



Die Steuerung

Die Steuerung ergibt sich als Moore-Maschine direkt aus den Informationen des Ablaufplans τ, der Bindung β,γ und der Registerbindung ρ.

Sie basiert auf einem rücksetzbaren Modulo L (Latenz) Zähler, bei dem mit jedem Zustand i, $0 \le i \le L-1$ folgende Ausgaben assoziiert sind:

- Für jede Kante (u,v) mit $\tau(v) = i$, $op(u,v) = j$:
  - ➔ gebe im Zustand i $selop(\beta(v),\gamma(v),op(u,v)) = Opindex[\beta(v),\gamma(v),op(u,v),\rho(u)]$ aus
- Für jeden Knoten v mit $\tau(v) + w(v) - 1 = i$:
  - ➔ gebe im Zustand i $selin(\rho(v)) = Rindex[\rho(v),\beta(v),\gamma(v)]$ aus

Damit lesen die Operationen im Startzyklus ihre Operanden aus den richtigen Registern, und die Register übernehmen in dem Zyklus, in denen eine Operation, die an das Register gebunden ist, endet, die neuen Werte.

Die Steuerung -- ff

Die Steuerung kann nun durch ein Synthesewerkzeug (vgl. Logiksynthese WS08/09) aus der Übergangstabelle erzeugt werden.

Problem:

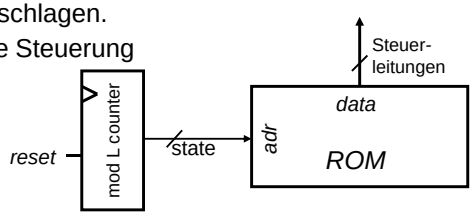
Die Moore Maschine zur Steuerung hat i.A. sehr viele Ausgabeleitungen, für die jeweils aus dem aktuellen Zustand eine Ausgabefunktion berechnet werden muss.

- ➔ komplexer Ausgabeschaltkreis

Alternative:

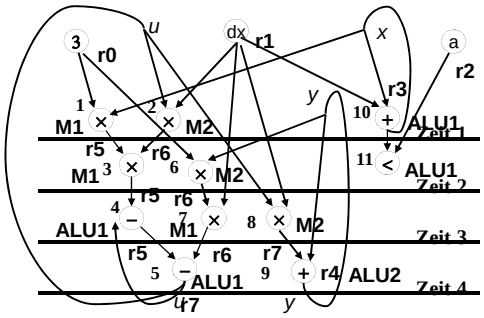
Benutze einen Zähler, und ein ROM der Wortbreite #Steuerleitungen um die Ausgabe nachzuschlagen.

- ➔ Mikroprogrammierte Steuerung



Implementierung: Beispiel

Wir betrachten den DGL Löser mit einem gültigen Ablaufplan für zwei Multiplizierer und 2 ALUs aus dem letzten Kapitel.
Wir hatten folgende Bindungen:



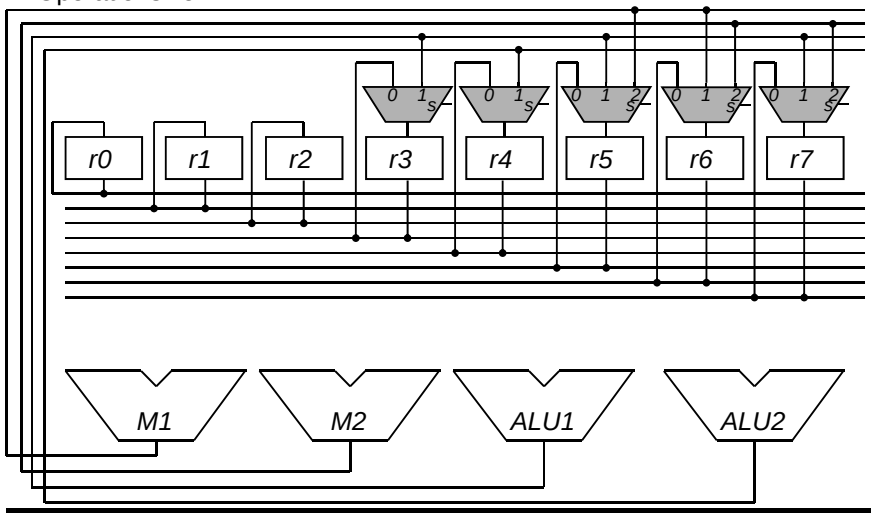
Input Register:
r0: → "3", r1 → dx, r2 → a
Gebundene Register:
r3: → Knoten 10
r4: → Knoten 9
r5: → Knoten 1,3 und 4
r6: → Knoten 2,6,7
r7: → Knoten 8,5

RES(r3) = { ALU1}
RES(r4) = { ALU2}
RES(r5) = { ALU1, M1}
RES(r6) = { M1, M2}
RES(r7) = { ALU1, M2}
OP(ALU1,1) = {r1,r3,r5,r7}
OP(ALU1,2) = {r2,r3,r5,r6}
OP(ALU2,1) = {r7}
OP(ALU2,2) = {r4}
OP(M1,1) = {r0,r5,r6}
OP(M1,2) = {r1,r3,r6}
OP(M2,1) = {r0,r7}
OP(M2,2) = {r1,r4}

Implementierung: Beispiel -- ff

Wir erhalten also folgendes Operationswerk:

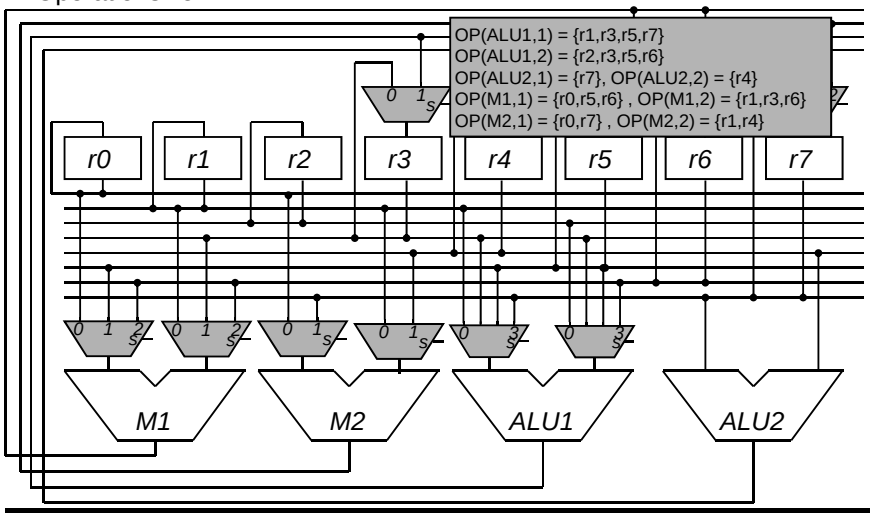
RES(r3) = { ALU1}
RES(r4) = { ALU2}
RES(r5) = { ALU1, M1}
RES(r6) = { M1, M2}
RES(r7) = { ALU1, M2}



Implementierung: Beispiel -- ff

Wir erhalten also folgendes Operationswerk:

OP(ALU1,1) = {r1,r3,r5,r7}
OP(ALU1,2) = {r2,r3,r5,r6}
OP(ALU2,1) = {r7}, OP(ALU2,2) = {r4}
OP(M1,1) = {r0,r5,r6} , OP(M1,2) = {r1,r3,r6}
OP(M2,1) = {r0,r7} , OP(M2,2) = {r1,r4}



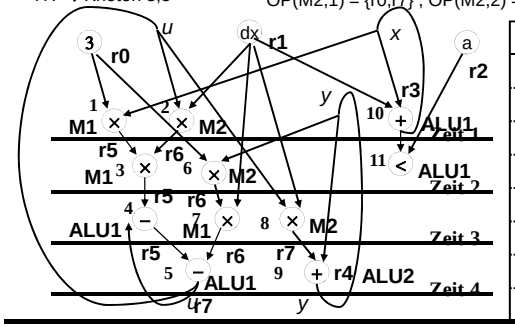
Implementierung: Beispiel -- ff

Zur Steuerung konstruiere man folgende Moore Maschine:

Input Register:
r0: → "3", r1 → dx, r2 → a
Gebundene Register:
r3: → Knoten 10
r4: → Knoten 9
r5: → Knoten 1,3 und 4
r6: → Knoten 2,6,7
r7: → Knoten 8,5

RES(r3) = { ALU1}, RES(r4) = { ALU2}
RES(r5) = { ALU1, M1}, RES(r6) = { M1, M2}
RES(r7) = { ALU1, M2}
OP(ALU1,1) = {r1,r3,r5,r7}
OP(ALU1,2) = {r2,r3,r5,r6}
// OP(ALU2,1) = {r7}, OP(ALU2,2) = {r4}
OP(M1,1) = {r0,r5,r6} , OP(M1,2) = {r1,r3,r6}
OP(M2,1) = {r0,r7} , OP(M2,2) = {r1,r4}

Ausgabeformat:
(selin(r3),…,selin(r7),
selop(ALU1,1),selop(ALU1,2),
selop(M1,1), selop(M1,2)
selop(M2,1), selop(M2,2))

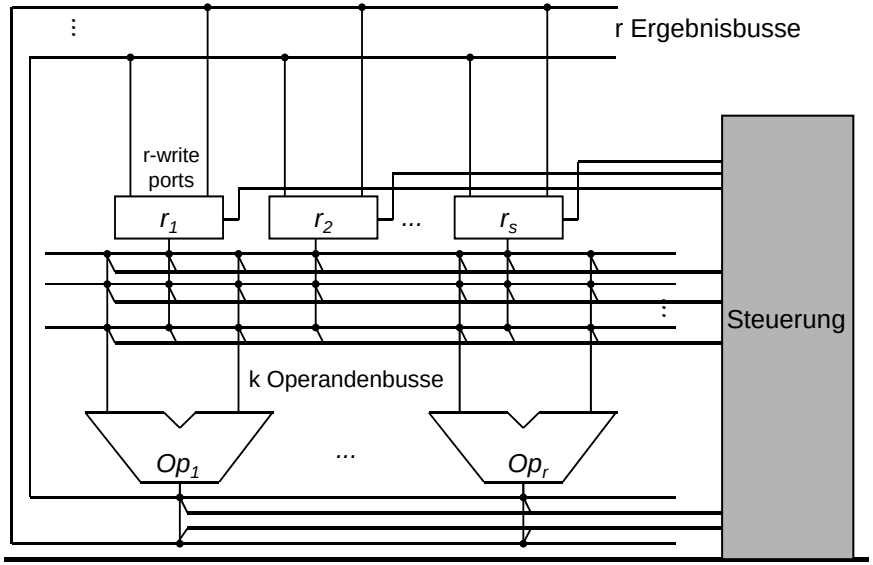


s	reset	δ(s,.)	λ(s)
0	0	1	1 0 2 2 0 0 1 0 1 1 0
1	0	2	0 0 2 2 0 1 0 1 2 0 1
2	0	3	0 0 1 1 2 3 2 2 0 1 0
3	0	0	0 1 0 0 1 2 3 0 0 0 0
0	1	0	1 0 2 2 0 0 1 0 1 1 0
1	1	0	0 0 2 2 0 1 0 1 2 0 1
2	1	0	0 0 1 1 2 3 2 2 0 1 0
3	1	0	0 1 0 0 1 2 3 0 0 0 0

Kritik

- Die Realisierung ist teuer! (Alle Multiplexer sind in entsprechenden Wortbreiten auszulegen.)
- Die Multiplexerdelays kommen zu den Verarbeitungszeiten hinzu, bei optimaler Auslegung im schlimmsten Fall *2 * log k* Stufen wo *k* der maximale Eingrad der Multiplexer ist.
- Die Halteoperation für Register könnte man auch in die Register integrieren und die Steuerung um ein *„write enable“* pro Register erweitern.
- Die Leistungsaufnahme ist hoch, weil wir stets auch inaktive Operationsschaltkreise an Register koppeln.
  - ⊙ Ausweg: ergänze die Operationsschaltkreise und die Steuerung um „enable“-Signale, die nicht genutzte Operationsschaltungen ihren alten Wert halten lassen.
- Wir haben noch keine Vorsorge getroffen, mehrere auf diese Weise automatisch erzeugte Komponenten systematisch zu einem System zu verschmelzen. (→ interessantes Forschungsthema!)

Busbasierte Implementierung



Kritik

- Die Realisierung ist in der Regel billiger als mit Multiplexern, da die Busse bzw. Registerports über Switches durchgeschaltet werden.
- Die Busdelays kommen zu den Verarbeitungszeiten hinzu!
- Man muss die Zahl der Busse nach dem maximalen Grad an Aktivität im Ablaufplan wählen, und zwar
  - *r* = max Zahl der produzierten Resultatsbits pro Zyklus
  - *k* = max Zahl der zu lesenden Operandenbits pro Zyklus

  oder die Ablaufplanung auf die Busbandbreite hin beschränken
- Die Steuerung bleibt teuer, weil man *k*#Register + k*#Operandeninputs + r*#Register + r*#Operationen* viele Steuerleitungen braucht. Dies können mehr werden als bei einer rein multiplexerbasierten Realisierung.

4.2.2 Zur Wahl der Taktperiode

Bei dieser Implementierungstechnik haben wir stillschweigend vorausgesetzt, dass alle Verabeitungszeiten $w(\beta(v))$ für die Ressource $\beta(v)$, an die ein Knoten v des Problemgraphens gebunden ist, als **Vielfaches der Taktperiode gegeben** sind und die zusätzlichen Verzögerungen durch die Multiplexer und die Steuerung in sich berücksichtigen.

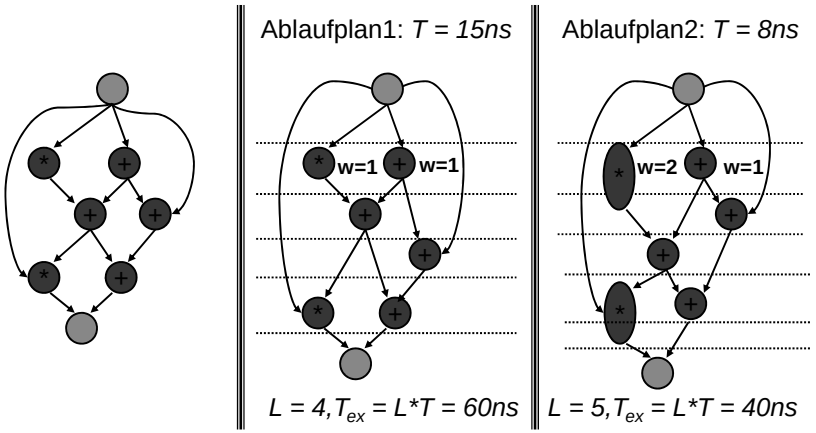
D.h. die Implementierung ist nur möglich für korrekte Ablaufpläne, die unter dieser Voraussetzung erzeugt werden.

Dass die Wahl der Taktperiode *T* einen entscheidenden Einfluss auf die durch die Ablaufplanung gegebene Latenz *L* und damit auf die effektive Bearbeitungszeit *T*L* hat, soll folgendes Beispiel verdeutlichen:

Wahl der Taktperiode -- ein Beispiel

Gegeben sei folgender Problemgraph, mit der Ressourcenbeschränkung 1 Addierer und 1 Multiplizierer.

Die Laufzeit des Multiplizierers sei 15ns, des Addierers 8ns.



Wahl der Taktperiode -- ff

Das Beispiel zeigt, dass je nach Wahl der Taktperiode andere Teilgraphen kritisch werden können. Im ersten Ablaufplan werden die 4 Additionen kritisch, da sie mit nur einem Addierer sequentiell ausgeführt werden müssen. Da diese zudem gegenüber der Taktperiode $T = 15ns$ einen Schlupf von $15ns - 8ns = 7ns$ haben, verliert man auf diesem Teilgraphen $4*7ns = 28\ ns$.

Im zweiten Ablaufplan ist der Pfad über 2 Multiplikationen und eine Addition kritisch. Dieser würde im asynchronen Fall, d.h. selbst wenn man ohne Rücksicht auf die Kosten den ganzen Problemgraphen durch einen einzigen Schaltkreis in einem Takt realisieren würde, schon $38ns$ Verzögerung haben. Da die Multiplizierer bei einer Taktperiode von $8ns$ nur einen Schlupf von $2*8ns - 15ns = 1ns$ haben, verliert man gegenüber dieser unteren Schranke sogar nur $2ns$.

Offenbar spielt also der Schlupf der Operationen unter T eine Rolle für die Effektivität von Ablaufplänen, die man für die Taktperiode T erstellen kann:

Wahl der Taktperiode -- ff

Offenbar genügt es auch, ausschließlich Taktperioden T zu betrachten mit $\delta(v)/T$ ganzzahlig für mindestens ein $v \in V_T$, denn

Annahme: $\delta(v)/T$ nicht ganzzahlig für alle $v \in V_T$

Dann setze

$$T' := max\left\{ \delta(v) \Big/ \left\lceil \frac{\delta(v)}{T} \right\rceil \;\middle|\; v \in V_T \right\}$$

Da $\delta(v)/T$ nicht ganzzahlig für alle $v \in V_T$, ist $T'<T$ und damit auch für alle $v \in V_T$

$$\left\lceil \frac{\delta(v)}{T'} \right\rceil \geq \left\lceil \frac{\delta(v)}{T} \right\rceil$$

Andererseits ist für alle v_j

$$\left\lceil \frac{\delta(v_j)}{T'} \right\rceil = \left\lceil \frac{\delta(v_j)}{max\left\{ \delta(v) \Big/ \left\lceil \frac{\delta(v)}{T} \right\rceil \middle| v \in V_T \right\}} \right\rceil = min\left\{ \frac{\left\lceil \frac{\delta(v)}{T} \right\rceil}{\delta(v)} \delta(v_j) \middle| v \in V_T \right\}$$

Wahl der Taktperiode -- ff

und $$min\left\{ \frac{\left\lceil \frac{\delta(v)}{T} \right\rceil}{\delta(v)} \delta(v_j) \middle| v \in V_T \right\} \leq \frac{\left\lceil \frac{\delta(v_j)}{T} \right\rceil}{\delta(v_j)} \delta(v_j) = \left\lceil \frac{\delta(v_j)}{T} \right\rceil$$

Also gilt für alle Ressourcen v_j, $\left\lceil \frac{\delta(v_j)}{T'} \right\rceil = \left\lceil \frac{\delta(v_j)}{T} \right\rceil$

Wir würden also das gleiche Planungsproblem erhalten bei kleinerer Taktperiode.

Definition -- mittlerer Taktschlupf

Unter einer Taktperiode T ist für eine Allokation α der **mittlere Taktschlupf** gegeben durch

$$mS(T) := \frac{\sum_{v \in V_T} \alpha(v) \cdot S(v,T)}{\sum_{v \in V_T} \alpha(v)}$$

Der Taktschlumpf

Minimierung des mittleren Taktschlupfes

Eine heuristische Möglichkeit, eine geeignete Taktperiode zu finden, besteht nun darin, für alle Taktperioden aus der Menge

$$\{\delta(v) \mid v \in V_T\},$$

eine Periode mit minimalem mittleren Taktschlupf zu suchen, und diese zu wählen.

In unserem Beispiel war der mittlere Taktschlupf

(7 + 0) / 2 ns = 3.5ns bei Wahl von $T = 15ns$,

und *(0 + 1) / 2 ns = 0.5 ns* bei Wahl von $T = 8ns$.

Allerdings ist dies nur eine einfache Heuristik, streng genommen müsste man für jeden Wert T aus $\{\delta(v) \mid v \in V_T\}$ ein Ablaufplanungsproblem unter Ressourcenbeschränkung lösen und sich dann für das T mit minimaler Bearbeitungszeit entscheiden.