Timing und Power

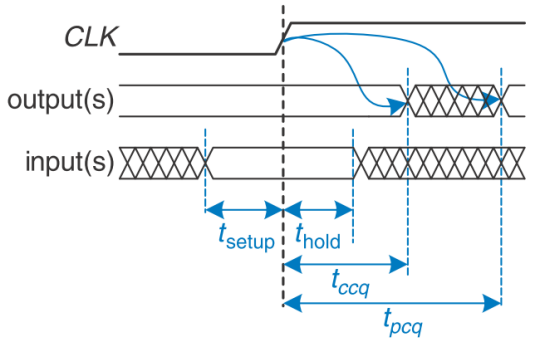
Timing - Einführung und Motivation

- Bisher haben wir uns auf die funktionale Spezifikation von synchronen (sequentiellen) Schaltungen fokussiert
- Wiederholung: Ein FF „kopiert“ den Eingang D zum Ausgang Q bei der steigenden Flanke („sampling D on the clock edge“)
- Wenn D stabil 0 oder 1 (bei der steigenden Taktflanke) ist, ist das Verhalten klar definiert
- Was passiert aber, wenn sich D bei der steigenden Flanke ändert?

- Vgl. Beispiel Kamera Blendenverschlusszeit (aperture time)
  - Wenn sich der zu fotografierende Bereich kurz vor und nach dem Auslöungszeitpunkt (aperture time) nicht still/stabil verhält (z.B. vorbeifahrendes Auto/Zug) wird das Bild unscharf

Timing - Definition wichtiger Zeiten

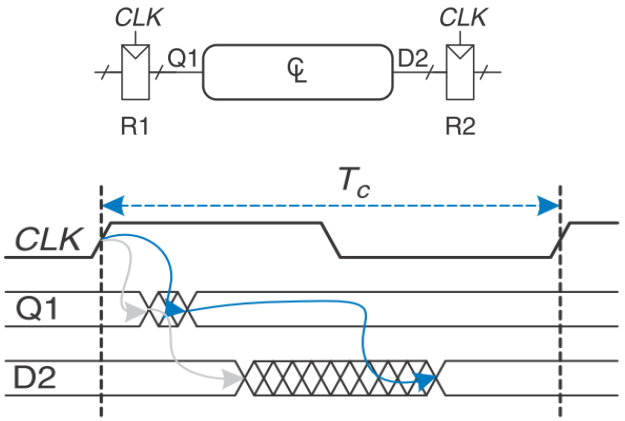
- Ein FF (synchrones sequentielles Element) hat auch eine solche aperture time, die sich aus der setup time (Tsetup) vor der Taktflanke und einer hold time (Thold) nach der Taktflanke zusammensetzt
  - Wenn diese aperture time verletzt wird, kann das Ausgangssignal auf 0 oder 1 kippen (glitch) oder für unbestimmte Zeit zwischen 0 und 1 hin und herschwingen



- Das clock-to-Q propagation delay Tpcq bestimmt dabei die Zeit, bis wann der Ausgang auf jeden Fall stabil ist
- Das clock-to-Q contamination delay Tccq bestimmt dabei die Zeit, ab wann der Ausgang sich nach einer Eingangsänderung frühestens beginnt zu ändern (aber noch nicht stabil ist)

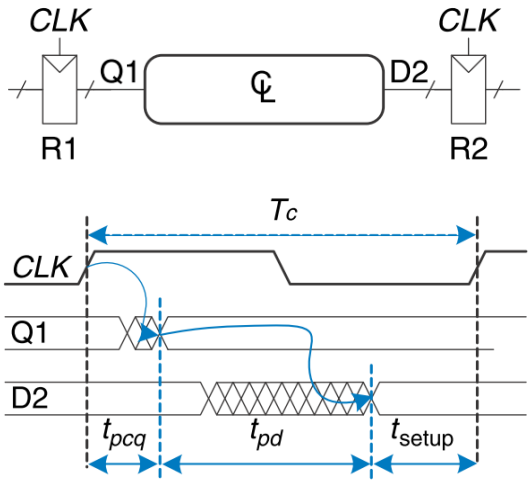
Timing - System timing - Taktfrequenz

- Die clock period oder cycle time Tc ist die Zeit zwischen zwei steigenden Flanken eines sich wiederholenden Taktsignals
- Der reziproke Wert Fc=1/Tc ist die Taktfrequenz
- Die Taktfrequenz wird in Hertz (Hz) oder Taktzyklen pro Sekunde gemessen:
  - 1MHz=1*10^6 Hz, 1GHz=1*10^9 Hz



Timing - System timing - Setup time constraint

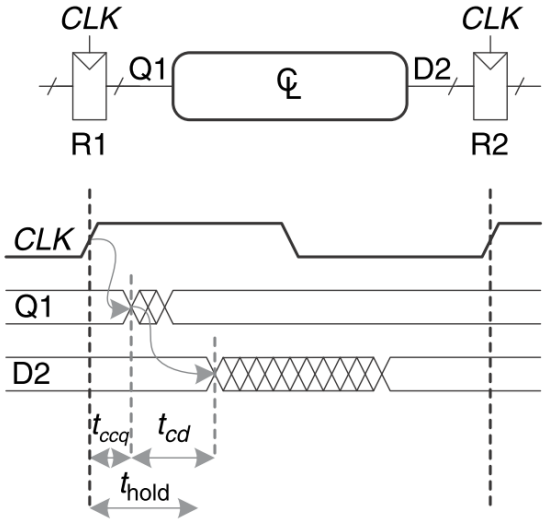
- Die blauen Pfeile zeigen das maximale Delay durch den kritischen Pfad
- Damit die Tsetup des FFs (hier R2) nicht verletzt wird, müssen die Ausgangsignale (hier D2) spätestens Tsetup vor der nächsten steigenden Taktflanke fertigberechnet sein (stabil sein)
- Die kleinstmögliche Taktperiode ist damit:
  - `Tc>=Tpcq+Tpd+Tsetup`
- Da Tc und Tpcq (HW-Hersteller) und Tsetup (Entwicklungsleiter/Marketing) meist vorgeben sind, wird nach Tpd umgestellt:
  - `Tpd<=Tc-(Tpcq+Tsetup)` //SETUP CONSTRAINT



- wenn Tpd vor Tsetup fertig wird (also vor der deadline) -> positiver slack (Pufferzeit)

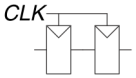
Timing - System timing - Hold time constraint

- Die grauen Pfeile zeigen das minimale Delay durch den kürzesten Pfad
- Damit Thold des FFs (hier R2) nicht verletzt ist, dürfen die Eingangssignale (hier D2) sich die Zeit Thold lang nach der steigenden Taktflanke nicht ändern
- D2 wird sich nach Tccq +Tcd ändern
  -> `Tccq+Tcd>=Thold`
- Da Tpcq und Thold meist vorgeben sind (HW-Hersteller), wird nach Tcd umgestellt:
  - `Tcd>=Thold-Tccq`
    //HOLD CONSTRAINT



Timing - System timing - Hold time constraint - Spezialfall zwei FFs hintereinander

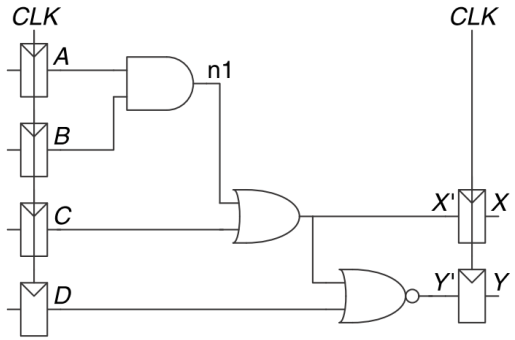
- Wie sieht das aus, wenn 2 FFs direkt hintereinander geschalten werden? Ist das möglich?



- Tcd>=Thold-Tccq: Tcd=0 da es keine kombinatorische Logik dazwischen gibt -> Thold<=Tccq
- Das heißt, verlässliche FFs müssen eine Thold haben die kürzer als ihr contamination delay Tccq ist
- Dies ist in der Praxis meistens gegeben, da man beim Design von FFs versucht die Thold von FFs möglichst auf Null zu setzen
- Verletzungen von Thold sind dadurch selten aber sehr kritisch, da sie sich (im Gegensatz zu Verletzungen von Tsetup) nicht einfach durch eine Anpassung von Tc lösen lassen
- Die einzige Lösung ist die Anpassung von Tcd, was heißt, dass die Schaltung neu designed werden muss!

Timing - System timing - Beispiel Timing Analyse I/II - Übung

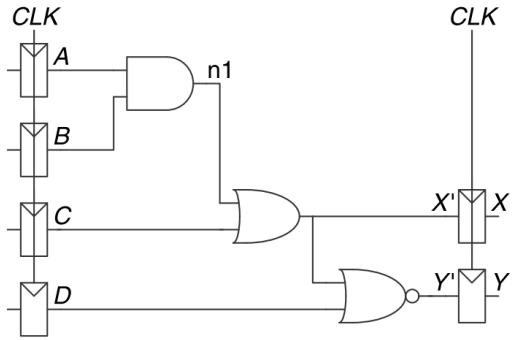
- Gesucht: Fmax
  - Fmax:



- Gegeben:
  - FFs: Tccq=30ps; Tpcq=80ps; Tsetup=50ps; Thold=60ps
  - Logik: für jedes Gatter: Tpd=40ps; Tcd=25ps

Timing - System timing - Beispiel Timing Analyse II/II

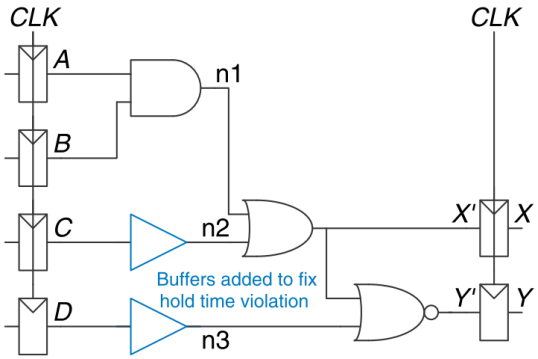
- Gesucht: Überprüfung setup constraint und hold constraint;
  - setup constraint: da bereits in Gleichung von Tc enthalten -> ok
  - hold constraint:
    - Kürzester Pfad ist wenn A=0 und C=0->1 für X‘ bei 1 Gatter
    - Tccq+Tcd>=Thold -> ?
      - Tccq+Tcd=30+25=55ps; Thold=60ps ->Verletzung HOLD CONSTRAINT



- Gegeben:
  - FFs: Tccq=30ps; Tpcq=80ps; Tsetup=50ps; Thold=60ps
  - Logik: für jedes Gatter: Tpd=40ps; Tcd=25ps

Timing - System timing - Beispiel Timing Analyse - Fixing hold time violation (B) (Q)

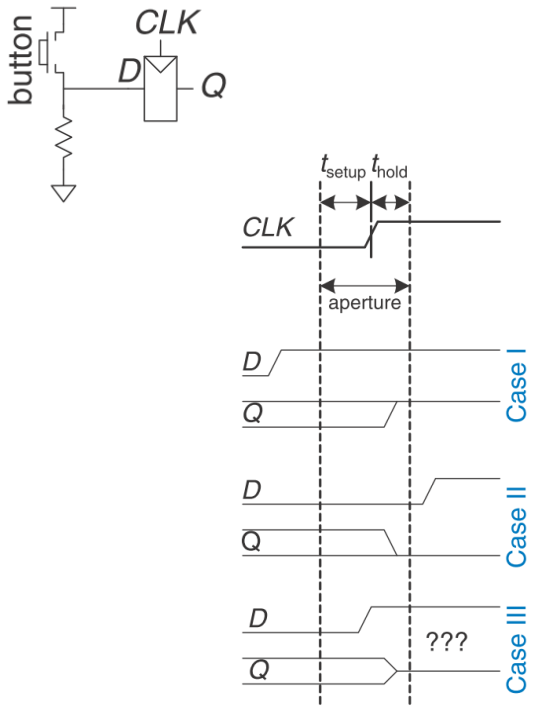
- Durch das Einfügen von Buffern verlängert man den kürzesten Pfad
- Gesucht: Überprüfung hold constraint;
  - hold constraint:
    - Kürzester Pfad ist wenn A=0 und C=0->1 für X‘ bei 1 Gatter + 1 Buffer
    - Tccq+Tcd>=Thold -> ?
    - Tccq+Tcd=30+25+25=80ps; Thold=60ps ->ok



- Gegeben:
  - FFs: Tccq=30ps; Tpcq=80ps; Tsetup=50ps; Thold=60ps
  - Logik: für jedes Gatter und Buffer: Tpd=40ps; Tcd=25ps

Timing - Metastability - Prinzip und Beispiel

- In der Realität ist es nicht immer möglich zu garantieren, dass ein Eingang zu einer synchronen sequentiellen Schaltung während der aperture time stabil ist
- Dies trifft insbesondere zu, wenn das Eingangssignal von außen kommt
- Beispiel: An ein FF angeschlossener button
  - Wenn button gedrückt -> D=1, ansonstden D=0
  - Fall 1 (wenn button gedrückt weit vor CLK) -> Q=1
  - Fall 2 (wenn button gedrückt weit nach CLK) -> Q=0
  - Fall 3 (wenn button gedrückt nach Tsetup und vor Thold /während aperture time) -> Ausgang ist nicht definiert (metastabil)

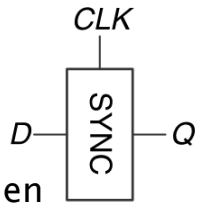


Timing - Metastability - Auswirkung

- Wenn ein FF metastabil ist, kann der Ausgang temporär Werte zwischen 0 und 1 (auch in der forbidden zone) annehmen
- Nach der unbestimmten resolution time Tres, wird der FF-Ausgang schließlich wieder einen stabilen Wert 0 oder 1 annehmen (welcher Wert ist aber unbestimmt)
- Theoretische und experimentelle Untersuchungen führen zu dem Ergebnis, dass die Wahrscheinlichkeit das Tres eine beliebige Zeit t überschreitet, exponentiell mit t sinkt
- $$P(t_{res} > t) = \frac{T_0}{T_c} e^{-\frac{t}{\tau}}$$
  - Tc= Taktperiode; To(FF charkteristisch)/Tc= Wahrscheinlichkeit, dass sich Eingangssignal zu einer nicht erlaubten Zeit ändert; τ=wie schnell sich FF aus metastabilen Zustand wegbewegt (FF charakteristisch, typ. Bruchteil einer ns);
    Die Gleichung gilt nur für t>>Tpcq

Timing - Synchronizers - Motivation, Zielsetzung und Eigenschaften

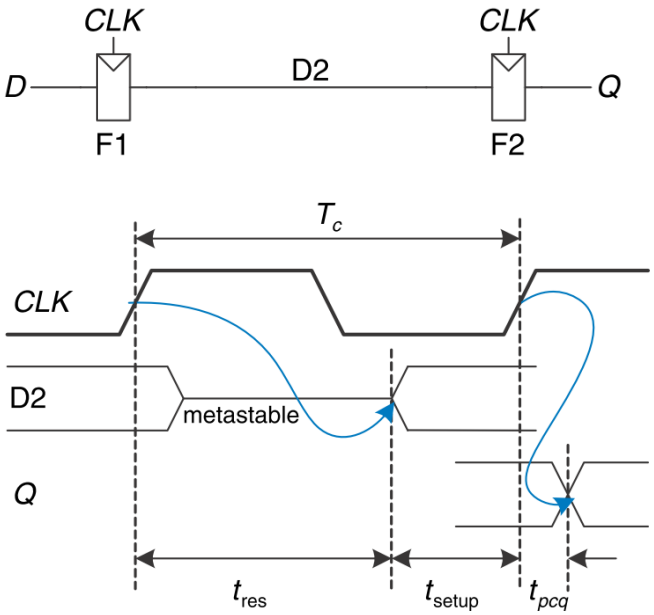
- Asynchrone Eingangssignale zu digitalen Systemen sind unvermeidbar
- Das Ziel ist, die Wahrscheinlichkeit, eines metastabilen Zustands ausreichend klein zu halten
- „Ausreichend klein“ hängt dabei vom Kontext ab:
  - Mobiltelefon: 1 Fehler in 10 Jahren ok (Benutzer startet Mobiltelefon neu)
  - Medizingerät: 1 Fehler in 10^10 Jahren besser (Universumsbegin)
- Um erlaubte Logik-Level zu garantieren, werden alle asynchronen Signale durch Synchronisierer (synchronizer) geführt



- Ein Synchronisierer hat einen asynchronen Eingang D und ein Takteingang
- Ein Synchronisierer erzeugt ein Ausgangsignal Q innerhalb einer bestimmten Zeit (mit gültigen Logik-Pegel und extrem hoher Wahrscheinlichkeit)
- Wenn D während aperture time stabil -> Q=D
- Wenn D während aperture time nicht stabil -> Q={1,0}, aber NICHT metastabil

Timing - Synchronizers - Prinzip

- Ein einfacher Synchronisierer kann aus 2 FFs aufgebaut werden
- F1 sampled D bei der steigenden Taktflanke
- Wenn D sich dabei ändert -> metastabil
- Wenn die Taktperiode lang genug ist, wird D2 sich mit hoher Wahrscheinlichkeit vor dem Ende der Taktperiode zu einem gültigen Logikpegel auflösen/einpendeln
- F2 sampled nun D2 (was nun stabil ist) und erzeugt somit einen gültigen Ausgangspegel



Timing - Pipelining - latency and throughput

- Die Geschwindigkeit eines Systems wird charakterisiert durch Latenz (latency) und Durchsatz (throughput)
- Ein token ist definiert als eine Gruppe von Eingangssignalen, die verarbeitet werden um eine Gruppe von Ausgangsignalen zu erzeugen
- Latenz= Zeit, die benötigt wird um einen token von Beginn bis Ende durch das System zu schleusen
- Durchsatz= Anzahl von token, die pro Zeit erzeugt werden können

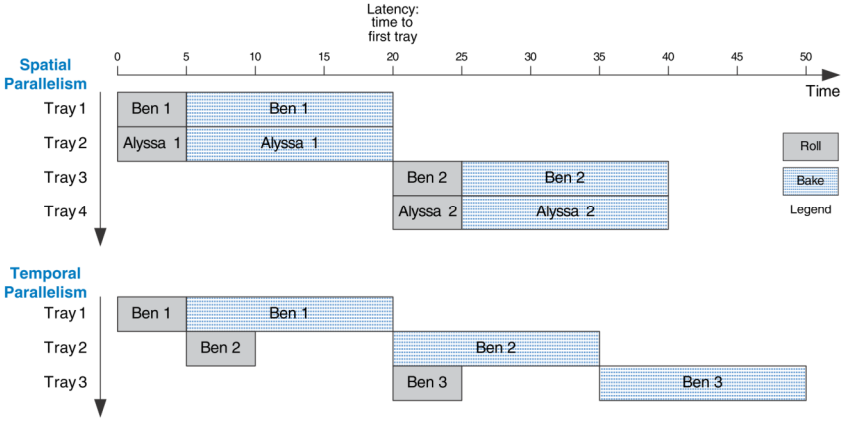
- Beispiel Keksbäckerei:
  - Ben backt Kekse für eine Party
  - Er braucht 5min um die Kekse zu rollen und auf ein Backblech (tray) zu platzieren
  - Es braucht 15min die Kekse im Ofen zu backen
  - Wenn dies fertig ist, beginnt Ben wieder von vorne mit dem nächsten Backblech
- F: Latenz, Durchsatz
- A: Ein Backblech mit Keksen ist ein token
  - Latenz=1/3h (pro Backblech)
  - Durchsatz= 3 Backbleche/h

Timing - Pipelining - Spatial and temporal parallelism

- Den Durchsatz kann man verbessern, indem man mehrere tokens gleichzeitig bearbeitet (parallelism)
- Man unterscheidet spatial parallelism und temporal parallelism
  - Bei spatial parallelism sind mehrere Kopien der Hardware vorhanden, so dass mehrere Aufgaben (tasks) gleichzeitig verrichtet werden können
  - Bei temporal parallelism werden die Aufgaben in kleinere Teilschritte (stages) aufgebrochen (wie bei der Fließbandfertigung)
  - Mehrere Aufgaben kann man gleichzeitig über mehrere Teilschritte verteilen
  - Jede Aufgabe muss durch alle Teilschritte geschleust werden, so dass zu einen beliebigen Zeitpunkt sich die Bearbeitung von mehrere Aufgaben in den Teilschritten überlappt
- Temporal parallelism nennt man auch pipelining

Timing - Pipelining - Spatial and temporal parallelism - Beispiel Keksbäckerei

- Beispiel Keksbäckerei
  - Ben hat Hunderte von Gästen und muss die Kekse nun schneller und effizienter backen
  - **Spatial parallelism**: Ben fragt seine Freundin Alyssa ihm zu helfen. Alyssa hat einen eigenen Ofen und ein eigenes Backblech
  - **Pipelining**: Ben kauft sich ein weiteres Backblech und rollt und platziert die Kekse auf dem zweiten Blech während das erste Blech im Ofen am Backen ist



- F: Latenz, Durchsatz
- A: Latenz (in allen Fällen)= 1/3h
  - Durchsatz spatial parallelism= 2* Durchsatz_alleine= 6 Backbleche/h
  - Durchsatz temporal parallelism= 4 Backbleche/h (Ben schiebt alle 15min ein Blech in den Ofen -> 4 Backbleche/h)
- Wenn man sowohl pipelining als auch spatial parallelism nutzt -> 8 Backbleche/h (2*Durchsatz_tempparalelism)

Timing - Pipelining - Spatial and temporal parallelism - summary

- Eine Aufgabe hat eine Latenz von L
- `Durchsatz_ohne= 1/L`
- `Durchsatz_spatialparallelism= N/L`
  //mit N-facher Hardware
- `Durchsatz_temporalparallelism= 1/(L/N)=N/L`
  //mit N-Teilschritten (gleicher Länge)
- `Durchsatz_temporalparallelism= 1/L1`
  //mit L1 als längsten Teilschritt

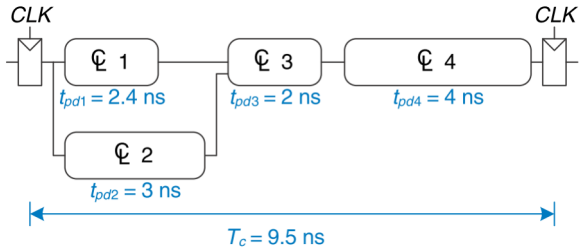
Timing - Pipelining - Pipelining for digital hardware

- Motivation und Prinzip

- Pipelining ist besonders interessant, da eine Geschwindigkeitsverbesserung ohne Vervielfachung der Hardware erreicht werden kann
- Register werden zwischen Blöcken von kombinatorischer Logik platziert, um die Logik in kürzere Teilschritte zu zerlegen, die schneller getaktet werden können
- Die Register verhindern, dass ein token den token im nächsten Teilschritt kaputtmacht/überschreibt

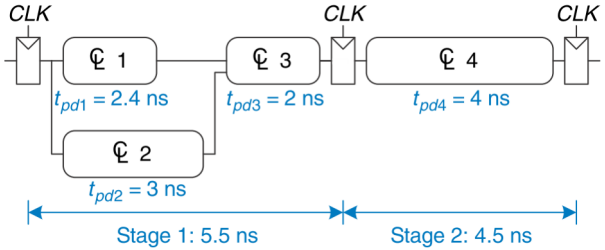
Timing - Pipelining - Pipelining for digital hardware - Beispiel mit 2-stufigen pipelining

- Beispiel Schaltung ohne pipelining



  - Geg: Tpcq=0.3ns; Tsetup=0.2ns
  - Ges: Latenz, Durchsatz
    - Latenz:
      - Kritischer Pfad = 2->3->4
      - Tc>= Tpcq+Tpd+Tsetup= 0.3+3+2+4+0.2=9.5ns
    - Durchsatz:
      - Durchsatz=1/Latenz= 1/9.5ns=105MHz

- Beispiel Schaltung mit 2-stufigen pipelining



  - Geg: Tpcq=0.3ns; Tsetup=0.2ns
  - Ges: Latenz, Durchsatz
    - Latenz:
      - Latenz Stufe 1: Tc>=Tpcq+Tpd+Tsetup= 0.3+2+3+0.2=5.5ns
      - Latenz Stufe 2: Tc>=Tpcq+Tpd+Tsetup= 0.3+4+0.2=4.5ns
      - !!! Der Takt muss langsam genug sein für beide -> Tc>=5.5ns -> Latenz=2*5.5ns=11ns
    - Durchsatz:
      - Durchsatz=1/L1=1/5.5ns=182MHz

Timing - Pipelining - Spatial and temporal parallelism - limitations

- Parallelism ist mächtig aber nicht in allen Situationen verwendbar
- Was parallelism verhindert sind (Daten-)Abhängigkeiten (dependencies)
- Wenn eine aktuelle Aufgabe von den Ergebnissen der hervorgehenden Aufgabe abhängig ist, kann die aktuelle Aufgabe nicht eher beginnen als die vorhergehende Aufgabe fertig ist
- Beispiel Keksbäckerei:
  - Ben will prüfen ob das erste Backblech der Kekse gut schmeckt, bevor er mit dem nächsten weiter macht
  - Diese Abhängigkeit verhindert spatial paralleliism und pipelining

Power - Motivation und Begriff

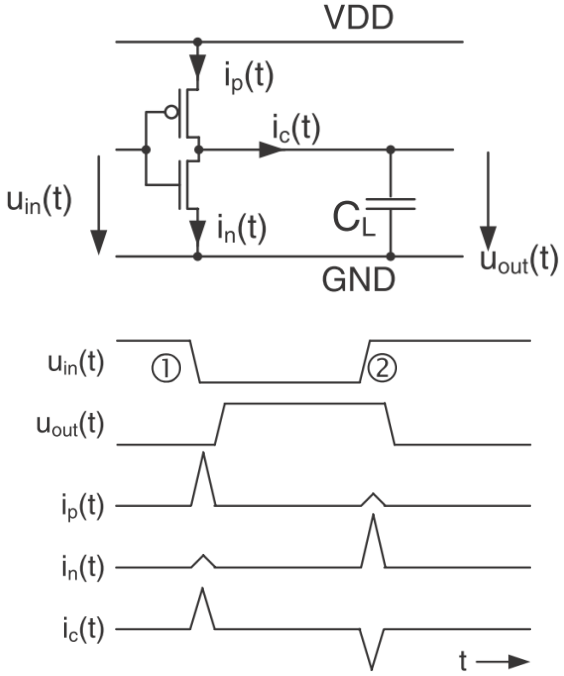
- Power consumption (Leistungsaufnahme)= Verwendete Energiemenge pro Zeiteinheit
- Leistungsaufnahme ist wichtig:
  - Bei batteriebetriebenen Geräten bestimmt die Leitungsaufnahme die Betriebszeit
  - Bei ans Stromnetz angesteckten Geräten bestimmt die Leitungsaufnahme die Betriebskosten durch Stromkosten und ggf. Kühlungskosten

- Die power consumption für CMOS besteht im Wesentlichen aus dynamic power und static power
  - 1. Dynamic Power= Leistung, die Kapazitäten für die Signalwechsel zwischen 0 und 1 umladen muss
  - 2. Static Power= Leistung, die trotzdem verbraucht wird, auch wenn es keine Signalwechsel gibt (idle mode)

Power - Dynamic power (Q)

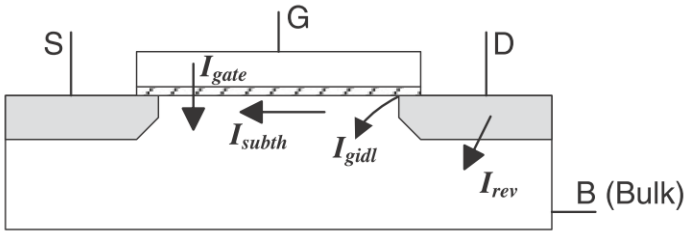
- Verdrahtungen und Logikgatter (Gateeingang) haben Kapazitäten
- Die Kapazität der Leitungen und die Kapazität der Gates der nachfolgenden Stufen werden durch den Kondensator Cl modelliert
- Die Energie, die benötigt wird, um eine Kapazität auf die Spannung Vdd zu laden ist `C*(Vdd)^2`
- Wenn die Kapazität mit der Frequenz f umgeschaltet wird (f mal pro Sekunde), wird die Kapazität f/2-Mal geladen und f/2-Mal entladen
- Das Entladen zählt nicht für die Leistungsaufnahme
- --> `Pdynamic=1/2*C*(Vdd)^2 *f`

- Beispiel CMOS Inverter



Power - Static power (Q)

- Elektrische Systeme ziehen selbst Strom wenn sie nichts machen und inaktiv sind
- Wenn Transistoren „off“ sind, sind sie nicht ganz aus, sondern ein ganz kleiner Strom fließt (leakage current)
- Der gesamte static current Idd (auch leakage current oder quiescent current genannt) fließt zwischen Vdd und GND
- -> `Pstatic=Idd *Vdd`



Power - Beispiel

- Mobiltelefon:
  Akku mit 6Wh (Watt Stunden) und 1.2 V; f=300MHz;
  C=10nF; Idd=40mA; Ptransmit=3W
- F: a) Betriebszeit wenn nicht genutzt (Tstandby),
  b) Betriebszeit wenn dauernd genutzt (Top)
- A:
  - a) Pstatic=Idd*Vdd=40mA*1.2V=48mW
    -> Tstandby=6Wh/48mW=125h
  - b) Pdynamic=1/2*C*(Vdd)^2 *f=
    1/2*10nF*(1.2)^2 *300MHz=2.16W
    Pges=Pstatic+Pdyn+Ptransmit=
    0.048W+2.16W+3W=5.2W
    -> Top=6Wh/5.2W=1.15h