

Timing und Power

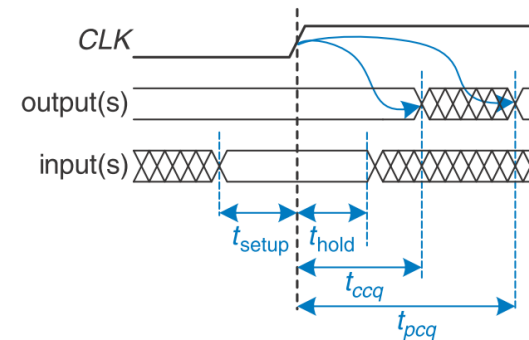
Timing - Einführung und Motivation

- 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 Auslösungszeitpunkt (aperture time) nicht still/stabil verhält (z.B. vorbeifahrendes Auto/Zug) wird das Bild unscharf

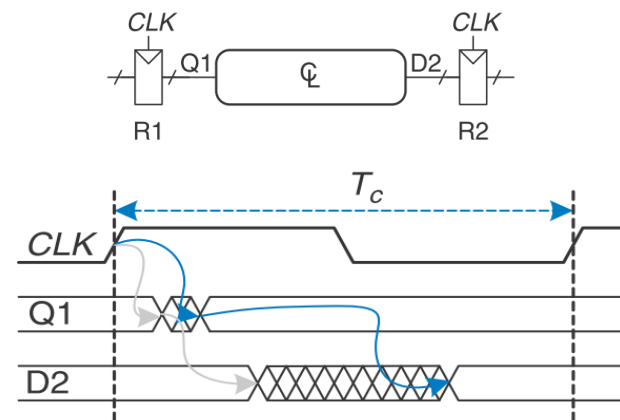
Timing - Definition wichtiger Zeiten

- Ein FF (synchrones sequentielles Element) hat auch eine solche aperture time, die sich aus der setup time (T_{setup}) vor der Taktflanke und einer hold time (T_{hold}) 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 T_{pcq} bestimmt dabei die Zeit, bis wann der Ausgang auf jeden Fall stabil ist
- Das clock-to-Q contamination delay T_{ccq} 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 T_c ist die Zeit zwischen zwei steigenden Flanken eines sich wiederholenden Taktsignals
- Der reziproke Wert $F_c = 1/T_c$ ist die Taktfrequenz
- Die Taktfrequenz wird in Hertz (Hz) oder Taktzyklen pro Sekunde gemessen:
 - $1\text{ MHz} = 1 \cdot 10^6\text{ Hz}$,
 - $1\text{ GHz} = 1 \cdot 10^9\text{ Hz}$



Timing - System timing - Setup time constraint

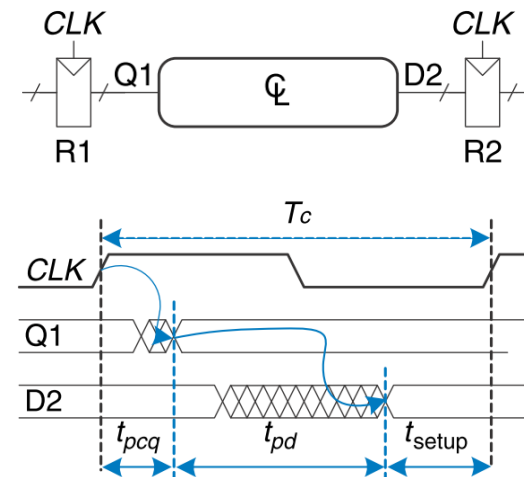
- Die blauen Pfeile zeigen das maximale Delay durch den kritischen Pfad
- Damit die T_{setup} des FFs (hier R2) nicht verletzt wird, müssen die Ausgangssignale (hier D2) spätestens T_{setup} vor der nächsten steigenden Taktflanke fertigberechnet sein (stabil sein)
- Die kleinstmögliche Taktperiode ist damit:

$$T_c \geq T_{pcq} + T_{pd} + T_{setup}$$

- Da T_c und T_{pcq} (HW-Hersteller) und T_{setup} (Entwicklungsleiter/Marketing) meist vorgeben sind, wird nach T_{pd} umgestellt:

$$T_{pd} \leq T_c - (T_{pcq} + T_{setup})$$

//SETUP CONSTRAINT

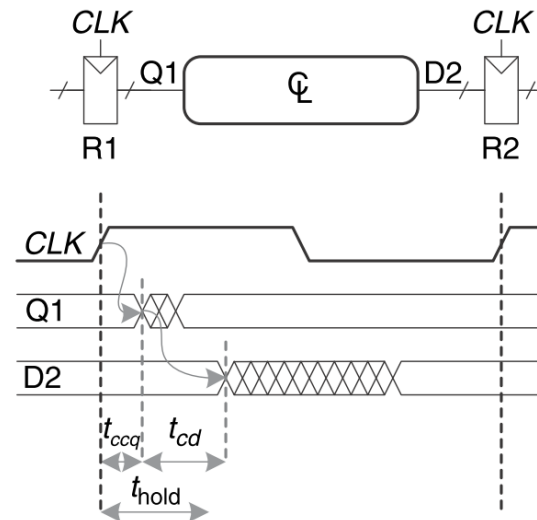


- wenn T_{pd} vor T_{setup} fertig wird (also vor der deadline) -> positiver slack (Pufferzeit)

entwickelte
Schaltung hat
Größenbeschr.
durch T_{pd}

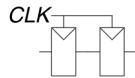
Timing - System timing - Hold time constraint

- 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 $T_{ccq} + T_{cd}$ ändern
-> $T_{ccq} + T_{cd} \geq T_{hold}$
- Da T_{ccq} und Thold meist vorgeben sind (HW-Hersteller), wird nach Tcd umgestellt:
▪ $T_{cd} \geq T_{hold} - T_{ccq}$
//HOLD CONSTRAINT



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

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



- $T_{cd} \geq T_{hold} - T_{ccq}$: $T_{cd} = 0$ da es keine kombinatorische Logik dazwischen gibt $\rightarrow T_{hold} \leq T_{ccq}$
- Das heißt, verlässliche FFs müssen eine **T_{hold}** haben die **kürzer als** ihr contamination delay **T_{ccq}** ist
- Dies ist in der Praxis **meistens gegeben**, da man beim Design von FFs versucht die T_{hold} von FFs möglichst auf Null zu setzen
- Verletzungen von T_{hold} sind dadurch **selten aber sehr kritisch**, da sie sich (im Gegensatz zu Verletzungen von T_{setup}) nicht einfach durch eine Anpassung von T_c lösen lassen
- Die **einzige Lösung** ist die Anpassung von T_{cd} , was heißt, dass die **Schaltung neu designed** werden muss!

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

- Gesucht: F_{max} $F_{max} = 1/T_c$
- F_{max} :

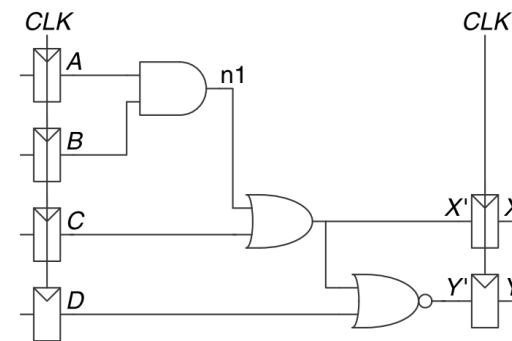
Kritischer Pfad ist wenn $B=1$, $C=0$, $D=0$ und $A=1 \rightarrow 1$ für Y' bei 3 Gattern

$$T_c \geq T_{pcq} + T_{pd} + T_{setup} =$$

$$T_{pcq} + 3 \cdot T_{pd} + T_{setup} =$$

$$80 + 3 \cdot 40 + 50 = 250 \text{ ps}$$

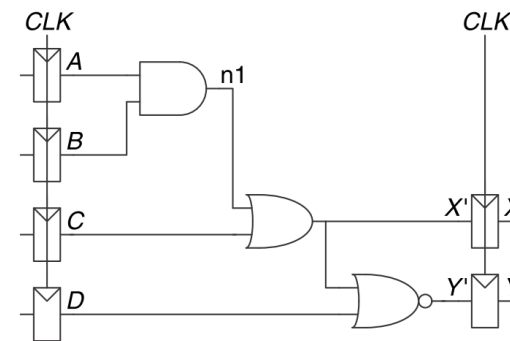
$$F_{max} = 1/T_c = 1 / 250 \text{ ps} = 4 \text{ GHz}$$



- Gegeben:
 - FFs: $T_{ccq}=30 \text{ ps}$;
 - $T_{pcq}=80 \text{ ps}$;
 - $T_{setup}=50 \text{ ps}$;
 - $T_{hold}=60 \text{ ps}$
 - Logik: für jedes Gatter:
 - $T_{pd}=40 \text{ ps}$; $T_{cd}=25 \text{ ps}$

Timing - System timing - Beispiel Timing Analyse II/II

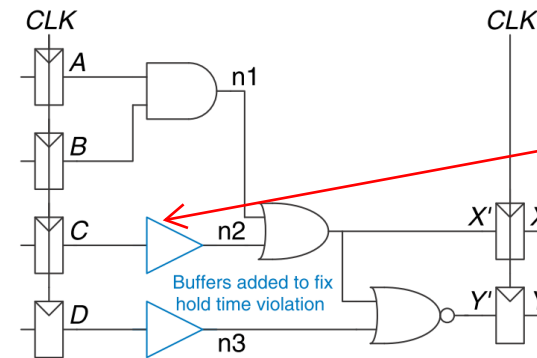
- Gesucht: Überprüfung setup constraint und hold constraint;
 - setup constraint: da bereits in Gleichung von T_c enthalten -> ok
 - hold constraint:
 - Kürzester Pfad ist wenn $A=0$ und $C=0 \rightarrow 1$ für X' bei 1 Gatter
 - $T_{ccq} + T_{cd} \geq T_{hold} \rightarrow ?$
 - $T_{ccq} + T_{cd} = 30 + 25 = 55\text{ps}$;
 - $T_{hold} = 60\text{ps}$
- > Verletzung HOLD CONSTRAINT



- Gegeben:
 - FFs: $T_{ccq} = 30\text{ps}$;
 - $T_{pcq} = 80\text{ps}$;
 - $T_{setup} = 50\text{ps}$;
 - $T_{hold} = 60\text{ps}$
 - Logik: für jedes Gatter:
 - $T_{pd} = 40\text{ps}$;
 - $T_{cd} = 25\text{ps}$

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 $C=0 \rightarrow 1$ für X' bei 1 Gatter + 1 Buffer
 - $T_{ccq} + T_{cd} \geq T_{hold} \rightarrow ?$
 - $T_{ccq} + T_{cd} = 30 + 25 + 25 = 80\text{ps}$;
 $T_{hold} = 60\text{ps} \rightarrow \text{ok}$

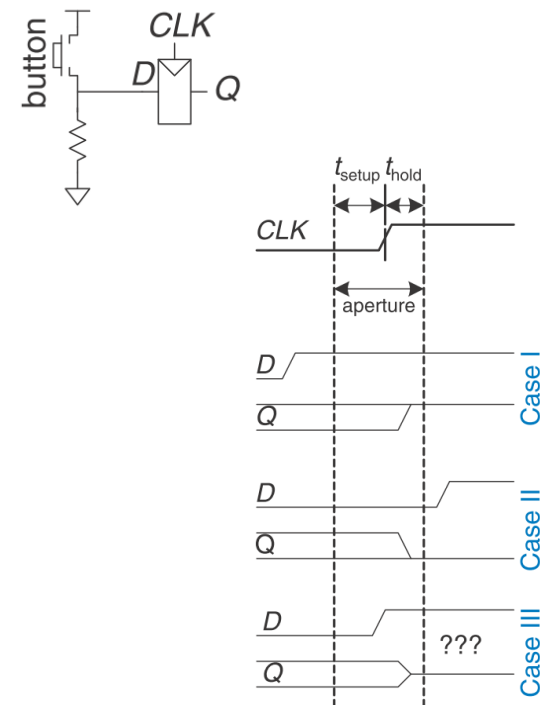


zus. Buffer erhöht
gesammt Tcd
(+25ps)

- Gegeben:
 - FFs: $T_{ccq}=30ps$; $T_{pcq}=80ps$; $T_{setup}=50ps$; $T_{hold}=60ps$
 - Logik: für jedes Gatter und Buffer: $T_{pd}=40ps$; $T_{cd}=25ps$

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 -> $D=1$, ansonsten $D=0$
 - 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 t_{setup} und vor t_{hold} /während aperture time) -> Ausgang ist nicht definiert (metastabil)



Timing - Metastability - Auswirkung

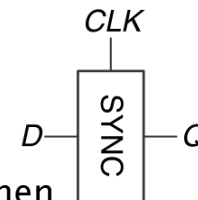
- Wenn ein FF metastabil ist, kann der Ausgang temporär Werte zwischen 0 und 1 (auch in der forbidden zone) annehmen
- Nach der unbestimmten resolution time T_{res} , wird der FF-Ausgang schließlich wieder einen stabilen Wert 0 oder 1 annehmen (welcher Wert ist aber unbestimmt)
- Theoretische und experimentelle Untersuchungen führen zu dem Ergebnis, dass die Wahrscheinlichkeit das T_{res} eine beliebige Zeit t überschreitet, exponentiell mit t sinkt

Herleitung nicht
relevant

- $P(t_{res} > t) = \frac{T_0}{T_c} e^{-\frac{t}{\tau}}$
- T_c = Taktperiode; T_0 (FF charakteristisch) / T_c = 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 $t \gg T_{pcq}$

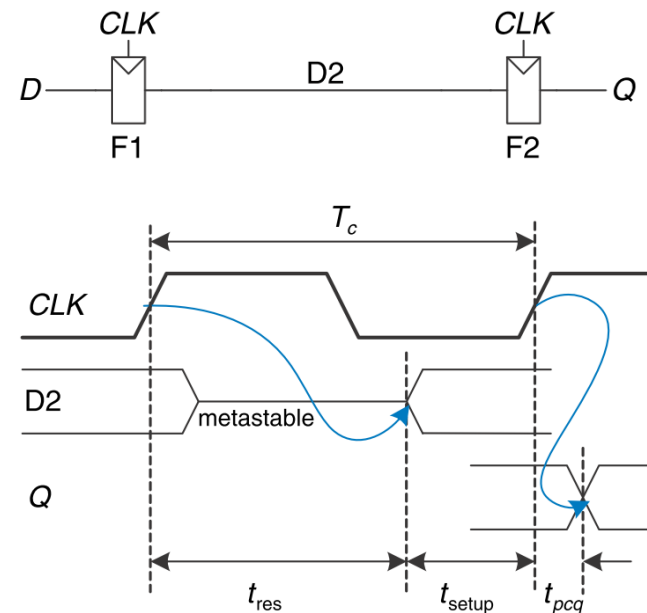
Timing - Synchronizers - Motivation, Zielsetzung und Eigenschaften

- 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 (Universumsbeginn)
 - 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 Ausgangssignal Q innerhalb einer bestimmten Zeit (mit gültigen Logik-Pegel und extrem hoher Wahrscheinlichkeit)
 - Wenn D während aperture time stabil $\rightarrow Q=D$
 - Wenn D während aperture time nicht stabil $\rightarrow Q=\{1,0\}$, 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



Asynchrone Signale können zur metastabilität führen
Lösen lässt sich das durch 2 FFs hintereinander

lang genug
warten bis Signal
sich stabilisiert zu
gültigem
Logikpegel

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 Ausgangssignalen 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

token = Backbleck

Timing - Pipelining - Spatial and temporal parallelism

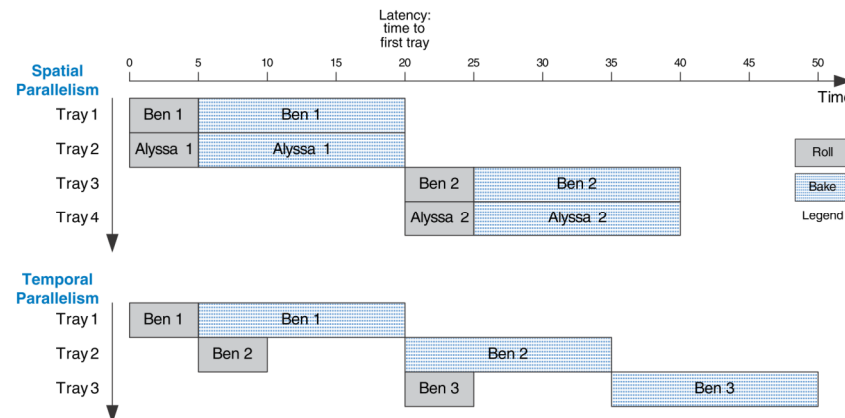
- Den Durchsatz kann man verbessern, indem man mehrere tokens gleichzeitig bearbeitet (parallelism)
- Man unterscheidet spatial parallelism und temporal parallelism
- Bei spatial parallelism sind mehrere Kopien der Hardware vorhanden, so dass mehrere Aufgaben (tasks) gleichzeitig verrichtet werden können
- 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 einem beliebigen Zeitpunkt sich die Bearbeitung von mehreren Aufgaben in den Teilschritten überlappt
- Temporal parallelism nennt man auch pipelining

mehere
Öfen/Bleche/Helfer



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 = $2 \times$
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
($2 \times \text{Durchsatz_tempparallelism}$)

bezogen auf
jedes Backblech

Timing - Pipelining - Spatial and temporal parallelism - summary

- Eine Aufgabe hat eine Latenz von L
- $\text{Durchsatz_ohne} = 1/L$
- $\text{Durchsatz_spatialparallelism} = N/L$
//mit N-facher Hardware
- $\text{Durchsatz_temporalparallelism} = 1/(L/N) = N/L$
//mit N-Teilschritten (gleicher Länge)
- $\text{Durchsatz_temporalparallelism} = 1/L_1$
//mit L_1 als längsten Teilschritt

Timing - Pipelining - Pipelining for digital hardware

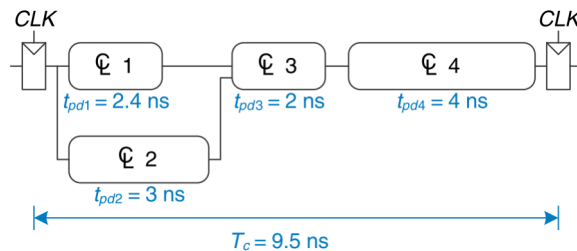
- Motivation und Prinzip

- 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

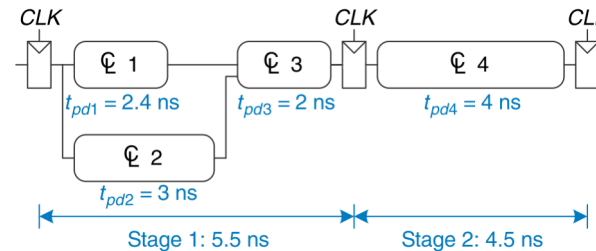
- Beispiel mit 2-stufigen pipelining

- Beispiel Schaltung ohne pipelining



- Geg: $T_{pcq} = 0.3 \text{ ns}$; $T_{setup} = 0.2 \text{ ns}$
- Ges: Latenz, Durchsatz
 - Latenz:
 - Kritischer Pfad = 2 \rightarrow 3 \rightarrow 4
 - $T_c \geq T_{pcq} + T_{pd} + T_{setup} = 0.3 + 3 + 2 + 4 + 0.2 = 9.5 \text{ ns}$
 - Durchsatz:
 - Durchsatz = $1 / \text{Latenz} = 1 / 9.5 \text{ ns} = 105 \text{ MHz}$

- Beispiel Schaltung mit 2-stufigen pipelining



- Geg: $T_{pcq} = 0.3 \text{ ns}$; $T_{setup} = 0.2 \text{ ns}$
- Ges: Latenz, Durchsatz
 - Latenz:
 - Latenz Stufe 1: $T_c \geq T_{pcq} + T_{pd} + T_{setup} = 0.3 + 2 + 3 + 0.2 = 5.5 \text{ ns}$
 - Latenz Stufe 2: $T_c \geq T_{pcq} + T_{pd} + T_{setup} = 0.3 + 4 + 0.2 = 4.5 \text{ ns}$
 - !!! Der Takt muss langsam genug sein für beide $\rightarrow T_c \geq 5.5 \text{ ns} \rightarrow \text{Latenz} = 2 * 5.5 \text{ ns} = 11 \text{ ns}$
 - Durchsatz:
 - Durchsatz = $1 / L1 = 1 / 5.5 \text{ ns} = 182 \text{ MHz}$

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 parallelism und pipelining

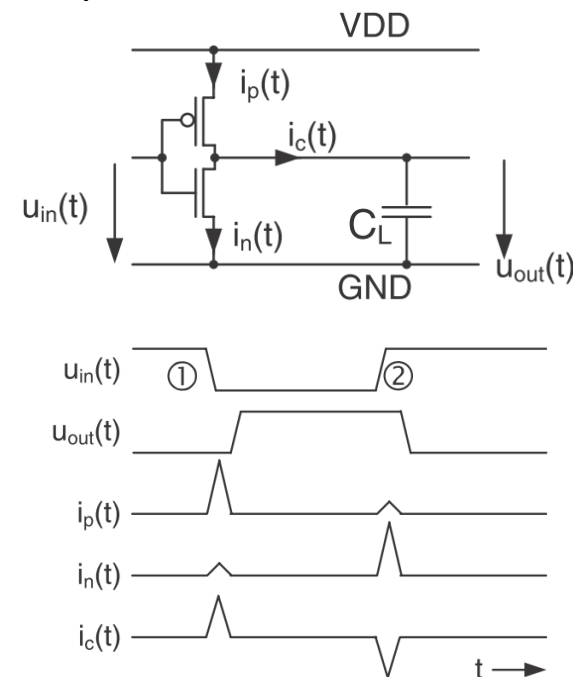
Power - Motivation und Begriff

- Power consumption (Leistungsaufnahme)= Verwendete Energiemenge pro Zeiteinheit
- Leistungsaufnahme ist wichtig:
 - Bei batteriebetriebenen Geräten bestimmt die Leistungsaufnahme die Betriebszeit
 - Bei ans Stromnetz angesteckten Geräten bestimmt die Leistungsaufnahme 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)

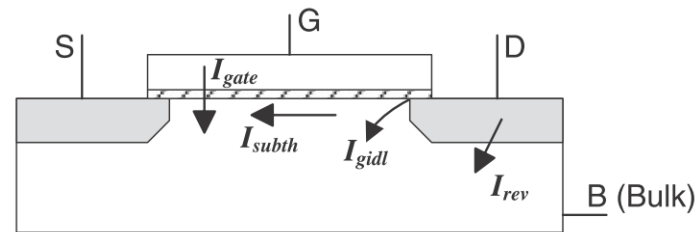
- Verdrahtungen und Logikgatter (Gateeingang) haben Kapazitäten
- Die Kapazität der Leitungen und die Kapazität der Gates der nachfolgenden Stufen werden durch den Kondensator C_L modelliert
- Die Energie, die benötigt wird, um eine Kapazität auf die Spannung V_{dd} zu laden ist $C * (V_{dd})^2$
- Wenn 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
- --> $P_{dynamic} = 1/2 * C * (V_{dd})^2 * f$

▪ 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 I_{dd} (auch leakage current oder quiescent current genannt) fließt zwischen V_{dd} und GND
- $\rightarrow P_{static} = I_{dd} * V_{dd}$



Power - Beispiel

- Mobiltelefon:
Akkumit 6Wh (Watt Stunden) und 1.2 V; $f=300\text{MHz}$;
 $C=10\text{nF}$; $I_{\text{dd}}=40\text{mA}$; $P_{\text{transmit}}=3\text{W}$
- F: a) Betriebszeit wenn nicht genutzt (T_{standby}),
b) Betriebszeit wenn dauernd genutzt (T_{op})
- A:
 - a) $P_{\text{static}}=I_{\text{dd}} \cdot V_{\text{dd}}=40\text{mA} \cdot 1.2\text{V}=48\text{mW}$
-> $T_{\text{standby}}=6\text{Wh}/48\text{mW}=125\text{h}$
 - b) $P_{\text{dynamic}}=1/2 \cdot C \cdot (V_{\text{dd}})^2 \cdot f=$
 $1/2 \cdot 10\text{nF} \cdot (1.2)^2 \cdot 300\text{MHz}=2.16\text{W}$
 $P_{\text{ges}}=P_{\text{static}}+P_{\text{dyn}}+P_{\text{transmit}}=$
 $0.048\text{W}+2.16\text{W}+3\text{W}=5.2\text{W}$
-> $T_{\text{op}}=6\text{Wh}/5.2\text{W}=1.15\text{h}$