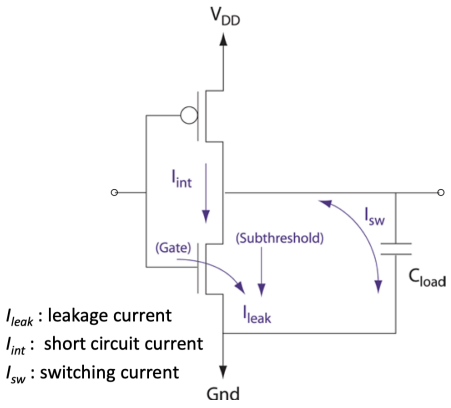
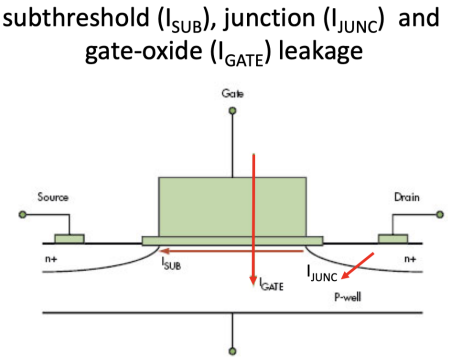
Embedded Systems - Notes Week 9

Ruben Schenk, ruben.schenk@inf.ethz.ch

January 16, 2022

We know look at the *power consumption* a CMOS gate. The following two diagrams show the different currents present in a CMOS gate:





The main sources for power consumption in CMOS processors are:

- *Dynamic power consumption:*
  - charging and discharging capacitors
  - short circuit power consumption
- *Leakage and static power:*
  - gate-oxide/subthreshold/junction leakage
  - becomes one of the major factors due to shrinking feature sizes in semiconductor technology

Power gating is one of the most effective ways of minimizing static power consumption (*leakage*). The idea is simple: cut-off power supply to inactive units and components.

We introduce the following two (simplified) relations:

1. Average power consumption of CMOS circuit (ignoring leakage):

$$P \sim \alpha C_L V_{dd}^2 f,$$

where:

- V_dd: supply voltage
- α: switching activity
- C_L: load capacity
- f: clock frequency

2. Delay of CMOS circuits:

$$\tau \sim C_L \frac{V_{dd}}{(V_{dd} - V_T)^2} \sim \frac{1}{V_{dd}},$$

where:

- V_{dd}: supply voltage
- V_T: threshold voltage ($V_T << V_{dd}$)

  **Dynamic Voltage Scaling (DVS): Decreasing V_{dd} reduces P quadratically (f constant). The gate delay increases reciprocally with decreasing V_{dd}. The maximal frequency f_{max} decreases linearly with decreasing V_{dd}.

In the end we have that:

$$E \sim \alpha C_L V_{dd}^2 ft = \alpha C_L V_{dd}^2 (\# \text{ cycles})$$

This leads to the final possible options for *saving energy* for a given task:

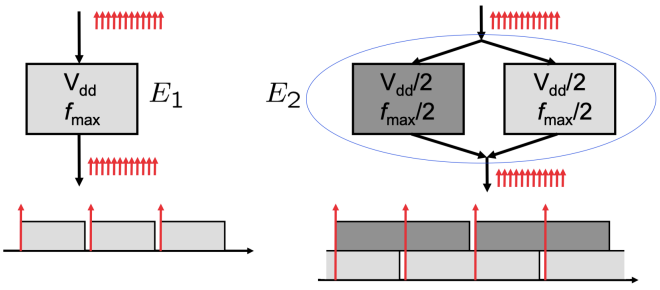
- reduce the supply voltage V_{dd}
- reduce switching activity α
- reduce the load capacitance C_L
- reduce the number of cycles # cycles

9.3 Techniques to Reduce Dynamic Power

We look at three different strategies to reduce the *dynamic power:*

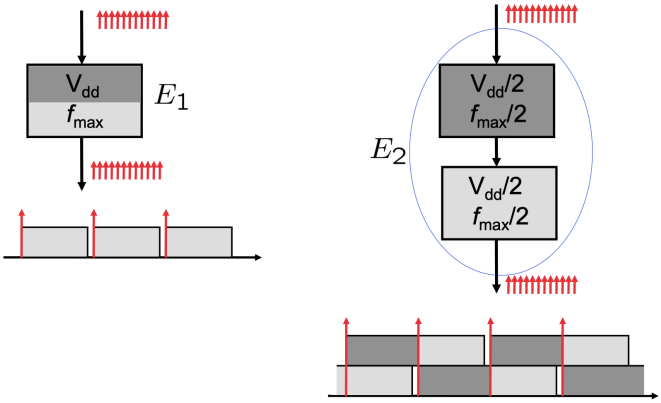
Parallelism:

- $E_1 \sim V_{dd}^2 (\# \text{ cycles})$
- $E_2 = \frac{1}{4} E_1$



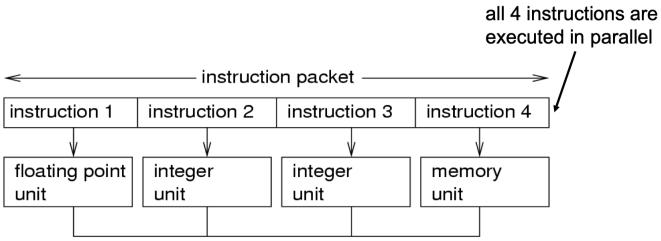
Pipelining:

- $E_1 \sim V_{dd}^2 (\# \text{ cycles})$
- $E_2 = \frac{1}{4} R_1$



Very Long Instruction Word (VLIW) Architectures:

- Large degree of parallelism
  - many parallel computational units, deeply pipelined
- Simple hardware architecture
  - explicit parallelism
  - parallelization is done offline



9.4 Dynamic Voltage and Frequency Scaling - Optimization

We quickly revisit the previously seen relations:

- $P \sim \alpha C_L V_{dd}^2 f$
- $E \sim \alpha C_L V_{dd}^2 ft = \alpha C_L V_{dd}^2 (\#\text{ cycles})$
- $f \sim \frac{1}{\tau} \sim V_{dd}$, where τ is the gate delay (?) and f is the maximum frequency of operation

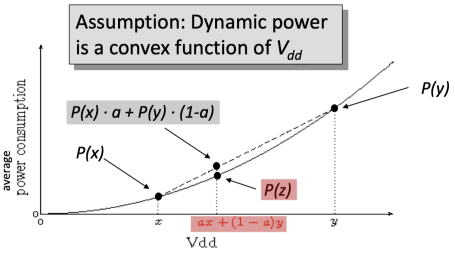
We look at different approaches in **dynamic voltage and frequency scaling (DVFS)** to find an optimal strategy. We first define:

- *gate delay:* $\tau \sim \frac{1}{V_{dd}}$
- *execution rate:* $f(t) \sim V_{dd}(t)$
- *invariant:* $\int V_{dd}(t)\, dt = \text{const.}$

Consider the following two cases:

- *Case A:* execute at voltage x for $T \cdot a$ time units and at voltage y for $(1-a) \cdot T$ time units. The *energy consumption* is $T \cdot (P(x) \cdot a + P(y) \cdot (2-a))$.
- *Case B:* execute at voltage $z = (P(x) \cdot a + P(y) \cdot (1-a))$ for T time units. The *energy consumption* is $T \cdot P(z)$.

This results in:



If possible, *running at a constant* frequency (voltage) minimizes the energy consumption for dynamic voltage scheduling. Case A is always worse if the power consumption is a convex function of the supply voltage.

Consider **real-time offline scheduling on one processor.** Let us model a set of independent tasks as follows: We suppose that a task $v_i \in V$:

- requires c_i computation time at normalized processor frequency 1
- arrives at time a_i
- has an absolute deadline constraint d_i

How do we schedule these tasks such that all these tasks can be finished no later than their deadlines and the energy consumption is minimized? We look at the **YDS algorithm.** Remember:

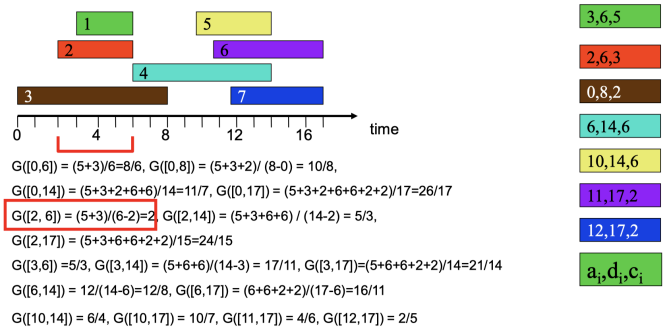
> If possible, running at a constant frequency (voltage) minimizes the energy consumption for dynamic voltage scaling.

YDS Optimal DVFS Algorithm for Offline Scheduling:

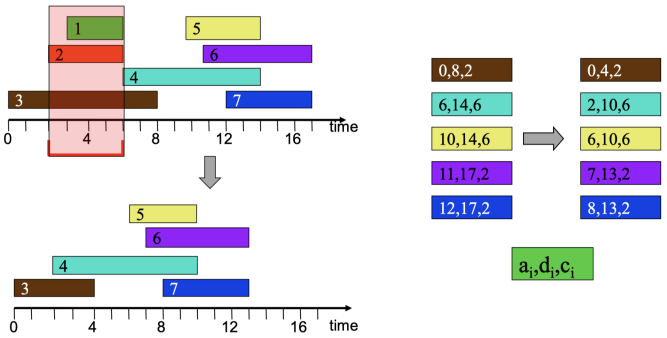
1. Define the **intensity** $G([z, z'])$ in some time interval $[z, z'$: average accumulated execution time of all tasks that have arrival and deadline in $[z, z']$ relative to the length of the interval $z - z'$:

- $V'([z, z']) = \{v_i \in V : z \leq a_i < d_i \leq z'\}$
- $G([z, z']) = \sum_{v_i \in V'([z, z'])} c_i/(z' - z)$

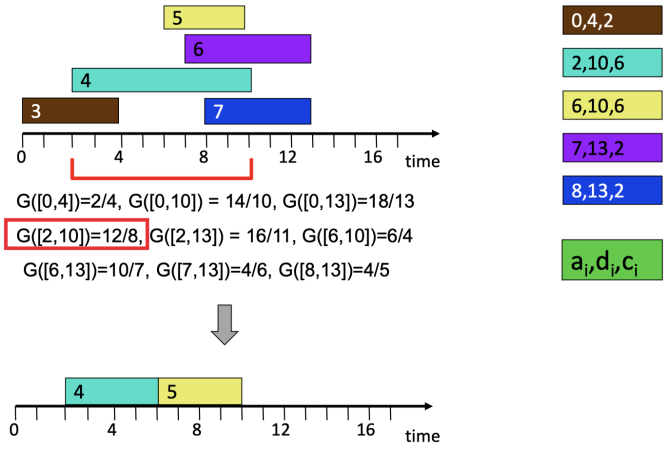
2. Execute the jobs in the interval with the highest intensity by using the earliest-deadline first schedule and running at the intensity as the frequency.



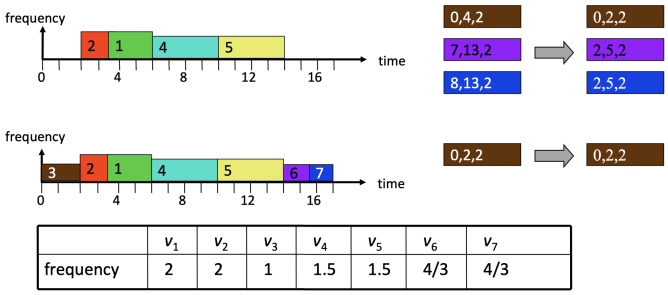
3. Adjust the arrival times and deadlines by excluding the possibility to execute at the previous critical intervals.



4. Run the algorithm for the revised input again.

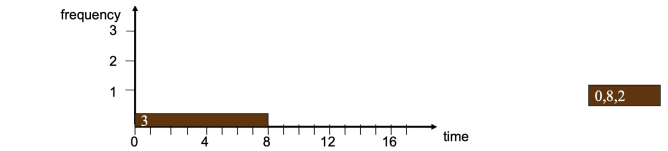


5. Put the pieces together.



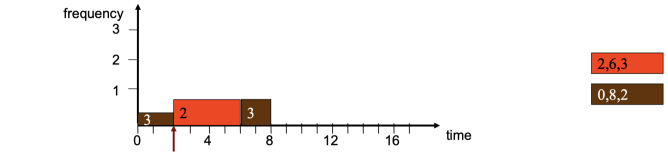
	v_1	v_2	v_3	v_4	v_5	v_6	v_7
frequency	2	2	1	1.5	1.5	4/3	4/3

YDS Optimal DVFS Algorithm for Online Scheduling: The idea is simple, we continuously update to the best schedule for all arrived tasks:



Continuously update to the best schedule for all arrived tasks:

Time 0: task v_3 is executed at 2/8

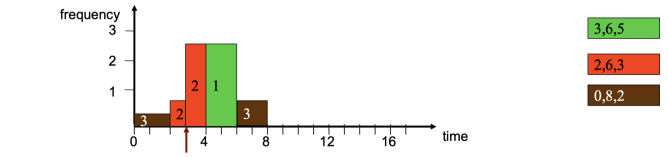


Continuously update to the best schedule for all arrived tasks:

Time 0: task v_3 is executed at 2/8

Time 2: task v_2 arrives

- G([2,6]) = ¾, G([2,8]) = 4.5/6=3/4 => execute v_8, v_2 at ¾



Continuously update to the best schedule for all arrived tasks:

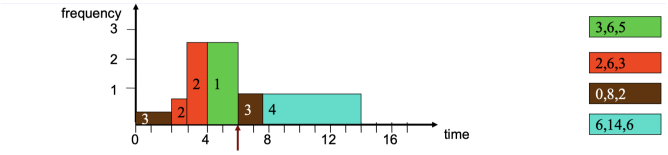
Time 0: task v_3 is executed at 2/8

Time 2: task v_2 arrives

- G([2,6]) = ¾, G([2,8]) = 4.5/6=3/4 => execute v_8, v_2 at ¾

Time 3: task v_1 arrives

- G([3,6]) = (5+3-3/4)/3=29/12, G([3,8]) < G([3,6]) => execute v_2 and v_1 at 29/12



Continuously update to the best schedule for all arrived tasks:

Time 0: task v_3 is executed at 2/8

Time 2: task v_2 arrives

- G([2,6]) = ¾, G([2,8]) = 4.5/6=3/4 => execute v_8, v_2 at ¾

Time 3: task v_1 arrives

- G([3,6]) = (5+3-3/4)/3=29/12, G([3,8]) < G([3,6]) => execute v_2 and v_1 at 29/12

Time 6: task v_4 arrives

- G([6,8]) = 1.5/2, G([6,14]) = 7.5/8 => execute v_3 and v_4 at 15/16

Remarks on the YDS Algorithm:

- Offline:
  - The algorithm guarantees the minimal energy consumption while satisfying the timing constraints.
  - The time complexity is $O(n^3)$, where n is the number of tasks in V
- Online:
  - Compared to the optimal offline solution, the online schedule uses at most 27 times of the minimal energy consumption.

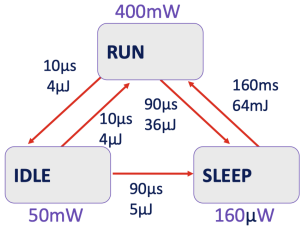
9.5 Dynamic Power Management

Dynamic power management (DPM) tries to assign the optimal power saving states during program execution. *Example:* StrongARM SA1100:

RUN: operational

IDLE: a SW routine may stop the CPU when not in use, while monitoring interrupts

SLEEP: Shutdown of on-chip activity

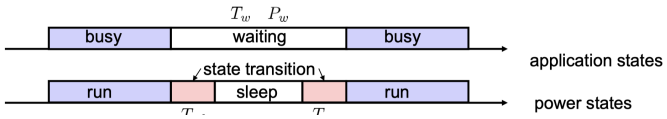


It is desired that a *shutdown* only happens during *long waiting times.* This leads to a trade-off between energy saving and overhead.

We define the **break-even time** as the minimum waiting time required to compensate the cost of entering an inactive (sleep) state. Entering an inactive state is beneficial only if the waiting time is longer than the break-even time. We make the following assumptions for the calculation:

- No performance penalty is tolerated
- An ideal power management that has the *full* knowledge of the future workload trace.

Consider the following two scenarios:



- Scenario 1 (no transition): $E_1 = T_w \cdot P_w$
- Scenario 2 (state transition): $E_2 = T_{sd} \cdot P_{sd} + T_{wu} \cdot P_{wu} + (T_w - T_{sd} - T_{wu}) \cdot P_s$

We can calculate the *break-even time* by limiting for T_w such that $E_2 \leq E_1$ which results in the following **break-even constraint**

$$T_w \geq \frac{T_{sd} \cdot (P_{sd} - P_s) + T_{wu} \cdot (P_{wu} - P_s)}{P_w - P_s},$$

and the following **time constraint:** $T_w \geq T_{ds} + T_{wu}$.