数逻笔记

November 2, 2024

1. 绪论

- 数字电路的优点：
  - 噪声不容易累积，稳定性好/可靠性高
  - 设计制造使用成本低
  - 便于传输存储
  - 自动化程度高
  - 更适合工艺演进
- CMOS 数字电路的功耗：
  - **静态功耗**：亚阈值漏电 (D-S)、栅极漏电 (B-G)、D/S 衬底漏电 (D-B, S-B)（极小）
  - **动态功耗**：对负载电容充放电产生的开关电流（占比最大）
  - **动态短路功耗**：PMOS 和 NMOS 短暂同时导通形成的短路电流
  - $E = \alpha C_{\text{Load}} V_{\text{DD}}^2 + I_{\text{off}} V_{\text{DD}} \tau + I_{\text{short}} V_{\text{DD}} \tau_{\text{0-1}}$, 降低功耗 $\rightarrow$ 降低电压 / 降低负载电容（改进工艺，缩小特征尺寸）/ 降低频率或翻转概率.
- 摩尔定律
  - 内容：集成电路的规模指数增长（每 1/1.5/2 年翻一番）
  - 意义：集成更多的晶体管，实现更高的性能、更低的成本和功耗
  - 限制：
    - **设计成本和周期的增加**
    - **量子不确定效应**
    - **功耗密度持续增加**
    - 工艺尺寸接近传统光刻机极限，新工艺成本过高
    - 寄生效应变得不可忽视
    - EDA 工具的支持尚不完善

2. 组合逻辑

- 补码：取反 + 1
  - 溢出的判定：正 + 负不溢出；正 + 正 / 负 + 负，最高位输出与最高位进位输出取异或.
  - 补码转十进制：首位负权值
- 卡诺图化简
  - 注意边缘、四角的相邻关系
  - 注意任意项取值
- 传输门/三态门，设计时避免高阻态.
- 噪声容限 $V_{\text{NH}} = V_{\text{OHmin}} - V_{\text{IHmin}}, V_{\text{NL}} = V_{\text{ILmax}} - V_{\text{OLmax}}$.
- 常用组合逻辑：$2^n - n$ 编码器 / $n - 2^n$ 译码器、$2^n : n$ 多路选择器 (MUX).
- 加法器
  - 行波进位加法器：全加器直接串联.
  - 超前进位加法器：无需等待 C_i 计算结束就能直接开始计算 C_{i+1}.
    - 进位传播 $P_i = A_i \oplus B_i$, 进位生成 $G_i = A_i \cdot B_i$
    - 递推式：$C_{i+1} = P_i \cdot C_i + G_i$.
    - 一般小于等于 4 位；超过四位：组内并行，组间并行.

3. 时序逻辑

- 时钟信号生成：环形振荡器
  - n 个非门（n 为奇数!），周期为 $n\left(t_{\text{pd,0-1}}+t_{\text{pd,1-0}}\right)$.
  - 两个点求与，可画图求解.
- SR 锁存器：$Q^+=S+R'Q$ $(SR=0)$
  - 若 $SR=1$, 当 S,R 同时变为 0 后输出振荡，因此需避免.
  - 门控 SR 锁存器：仅当使能 $C=1$ 时对输入透明, C 下降沿锁定.
  - $S=D, R=D'$, 变为 D 锁存器.
- D 触发器 / DFF
  - 边沿触发，只在触发沿透明.
  - 建立时间约束：$t_{\text{cycle}}\geqslant t_{\text{c-q}}+t_{\text{logic}}+t_{\text{su}}$.
  - 保持时间约束：$t_h\leqslant t_{\text{c-q}}+t_{\text{logic}}$.（一般原生满足）
- Mealy vs. Moore
  - 摩尔机：输出和状态变化同步，通常有更多状态（输出只与状态有关）;
  - 米利机：输出和状态变化异步，通常有更少状态（输出与状态和输入有关）.
- 状态化简
  - 行匹配：合并具有相同次态和输出的行
  - 蕴含表：（真有用吗?）
- 状态分配
  - 基于次态和输入/输出的准则
    - 最高优先级：输入相同的情况下，次态相同的现态
    - 中等优先级：同一状态的次态
    - 最低优先级：输出相同的现态
- 自启动分析
- 移位寄存器
- 计数器
  - 循环移位寄存器：环形计数器（$Q_0^+=Q_n$），扭环形计数器（$Q_0^+=Q_n'$）
  - 异步加法计数器：$Q_i^+=Q_i\oplus\text{IN}, \text{CLK}_{i+1}=Q_i'$.
  - 同步加法计数器：$Q_i^+=Q_i\oplus\prod_{\text{低位}}Q_k$

4. CPU

4.1. 单周期

4.2. 多周期

4.3. 流水线

- TODO
- 流水线冒险及解决方案
  - **结构冒险**：多条指令在同一个周期使用同一个硬件资源
  - **数据冒险**
    - 寄存器数据冒险
      1. Stall: 最朴素方法，造成吞吐率降低（3 个周期）;
      2. 寄存器堆下降沿写入，前半周期写，后半周期读（或：辅助旁路），减小 1 个周期;
      3. Forwarding:
         - EX/MEM.ALUout → ALU, MEM/WR.ALUout → ALU
         - 减小 2 个周期，可完全解决问题。

  - Load-use 冒险
    1. Forwarding: MDR(MEM/WR.) → ALU, 减小 1 个周期，仍需 Stall 1 个周期；
    2. 编译调度：调整指令顺序，在等待周期进行其他操作。这一步也可放在硬件。
  - **控制冒险**
    1. Stall 3 个周期
    2. Forwarding: 在 REG 阶段加入分支判断，减小 2 个周期，但可能造成时钟周期变长；
    3. 延迟槽：调整指令顺序，但需要编译器和 CPU 都配合（需要通知 CPU 无论如何都执行 beq 的下一条指令）；
    4. **分支预测**
       - 静态预测：总是预测跳转/总是预测不跳转，失败则撤销。
       - 动态预测：在 IF 阶段进行分支预测缓存
         - 额外的硬件支持：BHT(Branch Histoty Table) → BTB(Branch Target Buffer)
         - 查询指令地址是否在 BHT 和 BTB 中。若存在，根据历史记录判断是否跳转：若跳转，取出目标地址作为下一条指令的 IF 地址；若不存在，建立新的条目。
         - 下一条指令的 IF 阶段，根据预测目标地址取出指令。
- 数据冒险具体控制通路
  - 从 EX/MEM（即上一条指令）转发的条件：
    1. 前条指令需要写入寄存器（EX/MEM.RegWrite），且上一条指令写入的寄存器与当前要读取的寄存器一致（ID/EX.Rs[或 Rt] == EX/MEM.RegWrAddr）；(a)
    2. 前条指令写入的不是 $0（EX/MEM.RegWrAddr != 0）
       - 感觉可以表述成：当前指令要读取的不是 $0（ID/EX.Rs[或 Rt] != 0）$P_{-1} \wedge Q_0$
  - 从 MEM/WB（即前前条指令）转发的条件：
    1. 前前条指令需要写入寄存器，且与当前要读取的寄存器一致；
    2. 前前条指令写入的不是 0 号寄存器（MEM/WR.RegWrAddr != 0）；
       - 同样可以表述成：当前指令要读取的不是 $0（ID/EX.Rs[或 Rt] != 0）
    3. 不从前条指令转发（(a) 的非）。 $P_{-2} \wedge \neg P_{-1} \wedge Q_0$