Concepts

- gate-level pipelining

利用流水线的思想将处理过程分为多个阶段, 每个阶段执行不同的操作(比如指令获取, 译码, 执行, 写回),并将数据流从一个阶段传递到下一个阶段. 每个阶段被设计为同步的, 以便以恒定的时钟速度进行操作

- path balancing 在数字电路或芯片设计中, 为了确保信号在不同路径上具有相近的延迟或传输时间而采取的一种技术

信号需要通过多个逻辑门或线路来传递, 而每个逻辑门或线路的延迟可能会有所不同。如果在不同路径上的信号传输时间差异太大，可能会导致信号到达时序要求不一致，从而引发问题。 以下是一些常见的路径平衡技术：

1.链路插入（Buffer Insertion）：在长路径上插入缓冲器或寄存器，以减少路径的延迟，从而使其与其他路径相匹配。 2.等长线路（Equal-Length Paths）：在设计中，尽量确保所有信号的路径长度大致相等，以减少信号传播时间的差异。 3.时钟树设计（Clock Tree Synthesis）：在时钟分配中使用特殊的算法和布线技术，以确保时钟信号在整个芯片上的传播时间尽可能均匀。 4.时序优化工具：使用专门的时序优化工具来分析和调整设计，以满足时序要求并平衡路径延迟。 5.管理时序关系：设计者可以优化逻辑以减少关键路径上的逻辑门数量，从而减小路径延迟。

Topic

Design and implementation of bit-parallel RSFQ shift register(SR) memories

Report

这篇文章提出了一种基于RSFQ shift registers(SRs)的存储系统. 相比于vortex transitional(VT)memory cell(涡旋过渡存储单元), JJs(约瑟夫森结), SFQ&CMOS hybrid storage(混合存储)和ferromagnetic JJs(铁磁约瑟夫森结), RSFQ SRs的电路结构更为简单并且提供直流电. 并且, 由于与RSFQ-based processor的信号类型相同, 它可以很容易地与其集成在一起.

这是一组结构对比图, 对bit-serial memory来说, 在一个时钟周期内只能移动1bit的数据, 因此这种方式对于拥有更多位的数据来说就比较低; 而对于bit-parallel的方式来说, 它是由特定的D Flip Flop组成的列, 每个时钟周期能同时移动每一位数据, 效率更高.

这篇文章的主要工作首先是给出了一个8bit*2bit的bit-parallel memory结构图; 然后对于*8内存系统做一个仿真, 在SIMITNb03过程下评估最大频率和propagation delay传播延迟; 接着基于分层存储的概念, 提出8*8的内存可以作为一个block来构建更大的内存系统

这是在bit-parallel memory结构中的清除/写入以及读出数据的操作流程图. 首先对于clear操作来说, 起重要作用的是这个反馈环(feedback loop), 反馈环会在每一次数据移动后检测最右边这位是不是target data, 如果是, break; 对于write操作, data从最左边加入即可; 对于read操作, 当target data移动到最右边时, 不需要break反馈环, 从后面figure 3 可以看到需要读出的数据一个到output buffer, 读取数据; 另一条线经过反馈环最终到达原来的位置. 反馈环在这里的作用是保证读取的操作不会对原数据有影响(achieve nondestructive readout operations, NDRO)

这个是说对于读取bit-parallel memory中的数据所需要的时钟周期数取决于每个SR中的DFF的个数; 对于P*Q的内存系统来说, P代表SR的行数, Q代表每一行中(每个SR中)DFF的个数, 最后得到(1+Q)/2.

因为当第一个DFF写入数据之后, 总共需要Q个时钟周期这个数据才能到SR的最右端, 才能进入output buffer被读出, 然后被传回; 以此类推, 第1个DFF需要Q个cycle, 第Q个DFF需要1个cycle, 加起来即为结果

然后是这篇文章的一个重要的8bit*2bit的bit-parallel memory结构图

系统由四个主要部分组成。memory array用于数据存储，它由两个长度为八位的SR组成。SR中的内部反馈回路由无损读出（NDRO）逻辑门控制。input buffer用于临时存储输入数据，它由两个DFF对应两个SR组成。output buffer用于输出地址数据指定的一列数据，它由两个可复位 DFF （RDFF） 组成。

Controller由一个 3 转 8 的address decoder和一个 8 位 SR 组成。decoder由七个带互补输出（RDFFC）逻辑门的可复位DFF组成，用于电感优化。 互补输出指的是逻辑门具有正常输出和其反向（或互补）输出。例如，对于一个与门，如果正常输出是A AND B，那么它的互补输出可能是NOT (A AND B)。简单查了一下, 表示当电流通过它们时，它们存储在磁场中的能量的能力。随着数字电路以更高的频率（即更快的速度）运行，电感可能导致开关噪声和其他不希望的效应，这可能会干扰电路的正常运行。所以需要电感优化 控制器根据地址数据的解码结果在指定的特定时钟周期内产生触发输出控制信号。trigger out信号分为三条路径。 路径 1 和路径 3 分别input buffer和output buffer，路径 2 控制每个反馈环路中的 NDRO2。

clear/write以及readout操作和之前的相同 对于读取操作，使用 DFF 的第 8 列作为示例。在第一个时钟脉冲到达后，触发输出信号产生并通过由读取信号选通的路径3，以改变输出缓冲阵列中RDFF的状态，以便由第一个时钟脉冲转移到它们的数据作为存储器的输出发送出去。在第二个时钟脉冲引起的数据到达之前，必须清除RDFF的状态。用于清除的信号由触发输出信号生成，具有一些约瑟夫森传输线延迟(JTL delay)，以确保在数据读出后进行清除过程。取消选择信号将在每次读/写操作后发送以重置选择。 对于第八列DFF没有返回到原来的位置的情况, 原来存储在第八列的数据也会发生错误, 检查第八列的数据就可以知道错误发生, 使用几个时钟脉冲来纠正

这里提到了CB tree和时钟抖动(timing jitter)的概念, CB tree（Clocked Barrier tree）是用于快速同步超导电路中的Josephson Junction（JJ）开关活动的树状网络。 CB tree 的基本思想是提供一个结构，可以在一个特定的时钟周期内确保所有的信号都能同步地到达目标地点，从而允许电路中的各个部分同步工作。 "timing jitter"（时钟抖动）是一个描述在数字信号传输、处理或同步中，时钟或数据信号的不确定性或变异的术语。在电路设计中，jitter可以是由于多种原因产生的，包括热噪声、电源变化、外部干扰等。 "accumulation of timing jitter"（时钟抖动的累积）是指随着时间的推移或在信号路径上的连续阶段，jitter逐渐累积，使得总的不确定性或变异更加显著。 当电路中的某些部分（如解码器）被时钟路径穿越时，每次穿越都可能引入一些额外的jitter。随着时间的推移，这些jitter可能会累积，从而影响电路的性能和可靠性。

在此图中，ad1–ad3 表示从低到高的地址信号。地址“000”对应于内存阵列的第一列，“111”对应于 DFF 的第 8 列。 图中显示了九个步骤（I-IX）。信号sl和ds用于芯片选择：ds代表取消选择，用于防止时钟输入存储器 步骤 I 写入数据“11”以地址“000”。我们首先输入清除信号cr以清除控制器中解码器的状态，并发送地址，触发，写入和数据输入。然后，我们发送八个时钟脉冲来写入数据。步骤 III 显示地址 '000' 的读取操作。我们输入读取信号rd，数据“11”在第8个时钟周期输出，显示操作正确。

完全正确操作的直流偏置裕度经测试为±5%，但模拟裕度为±22%，该减小是由制造工艺引起的电路参数变化引起的。

直流偏置裕度是指当电源电压或温度变化时，电路仍能正常工作所需的最小电压或温度变化。

为什么直流偏置裕度通常比模拟裕度小： 温度和供电电压：电源电压和温度的变化对电晶体管的直流特性有很大的影响。随着电源电压的下降或温度的上升，电晶体管的驱动能力会减弱，从而影响电路的直流性能。而在模拟性能方面，这些变化的影响相对较小。 设计考虑因素：在设计电路时，直流裕度通常是首先考虑的，因为它对电路的整体性能有很大的影响。而模拟裕度通常是在电路已经满足直流性能要求后再考虑的。

对于 SIMIT-Nb03 工艺，在 34 GHz 最大工作频率下工作的存储器的直流偏置裕度被模拟为 ±16% ;具有更多Nb层(含铌的超导材料)的先进工艺可以帮助减少内存面积并降低功耗。

流水线设计是一种提高处理器吞吐量的方法，通过将指令的执行分解成多个阶段并使它们并行执行，可以在同一时刻处理多个指令的不同阶段。

当我们谈论"深"流水线时，我们指的是流水线拥有许多这样的阶段。例如，一个简单的流水线可能只有几个阶段（例如取指、解码、执行），而一个深流水线可能有十几个或更多阶段。

深流水线对并行访问产生延迟的原因：

增加的流水线阶段：每增加一个流水线阶段，都可能增加一个时钟周期的延迟。因此，深流水线设计可能需要更多的时钟周期来完成一个指令。

数据冒险：在并行执行的指令之间，如果一个指令依赖于另一个指令的结果，这可能导致数据冒险。深流水线可能增加这种数据依赖性，因为更多的指令在流水线中同时处理。

分支预测失误：流水线中的指令经常涉及到分支（例如if-else语句）。深流水线设计中，当分支预测失误时，可能需要清空流水线中的多个阶段，从而导致延迟。

资源争用：当多个指令并行执行并试图访问同一资源（例如内存或寄存器）时，可能会出现资源争用。深流水线可能会加剧这种争用，从而增加延迟。

为了应对这些挑战，现代处理器使用各种高级技术，如动态指令重新排序、乱序执行和复杂的分支预测算法，来减少深流水线中的延迟并提高性能。

这篇文章采用了分层存储的概念来扩展位并行存储器的容量，以8×8存储器作为构建块。

系统设计: 以M×N系统为例，系统由M/8个等级组成，每个等级包括N/8个8 × 8的内存块，对应于N位的数据宽度。一旦选择了一个等级，其他等级必须保持空闲。从一个等级的所有块中同时选择一个列来进行写入或读取

性能:表2列出了位并行系统的访问时间和带宽的估计。 为了扩展容量，通过增加等级的数量或增加等级中的块的数量来实现位并行内存，而不是增加SR的长度。 除了布线的增加外，访问所需的时钟周期和频率限制是恒定的。 由于基于8 × 8的块，其带宽比具有相同容量的位串行内存高。