

ECE 497NC:
Unconventional Computer Architecture

Lecture 20:
Reversible Computing

What is Reversible Computing?

- Basically, what it sounds like
 - Conventional computations run in one direction
 - Can't get back from the results of a computations to its inputs
 - Reversible computations can be run both forward and backward
 - Can regenerate inputs from outputs
 - Can stop at any point and back up

Why is this Interesting?

- **Power Consumption**
 - Original belief: Computation must consume at least $kT \cdot \ln(2)$ joules per bit computation
 - Revised analysis: only “logically irreversible” operations must consume energy
- **Reliability/Verification**
 - If a computation is reversible, you can back it up when you detect an error and determine exactly where the problem first occurred.
- **Quantum Computing**
 - For reasons we’ll go into later, basic operations in quantum computing must be reversible

Power Consumption – Theoretical Limits?

- **Basic Axioms/Previous Results**
 - Reversible computations have zero energy cost
 - Irreversibly provided/deleted bits have fixed cost
 - Reversible computations that generate y from x do not irreversibly delete y or x
- **Definitions**
 - X = input to computation
 - Y = output of computation
 - P = additional records in memory at start of computation (assumed to be deleted during computation) (such as the program description)
 - Q = garbage (temp) bits created during computation
- **Result**
 - Minimum energy to compute y given x is approximately $E(x,y) = K(x|y) + K(y|x)$
 - Intuitively, the size of the shortest program that computes y given x plus the size of the shortest program that computes x given y

Making Computation Reversible

- Many Basic Operations Destroy Data
 - $X = Y$ or Z
- Standard Memory Model Destroys Data
 - St r1, r2 overwrites old value in memory
 - Swap r1, r2, r3 preserves all data
- Need to address reversibility at multiple levels
 - Algorithm
 - Basic Operations
 - Circuit

Reversible Circuitry For Low Power Consumption

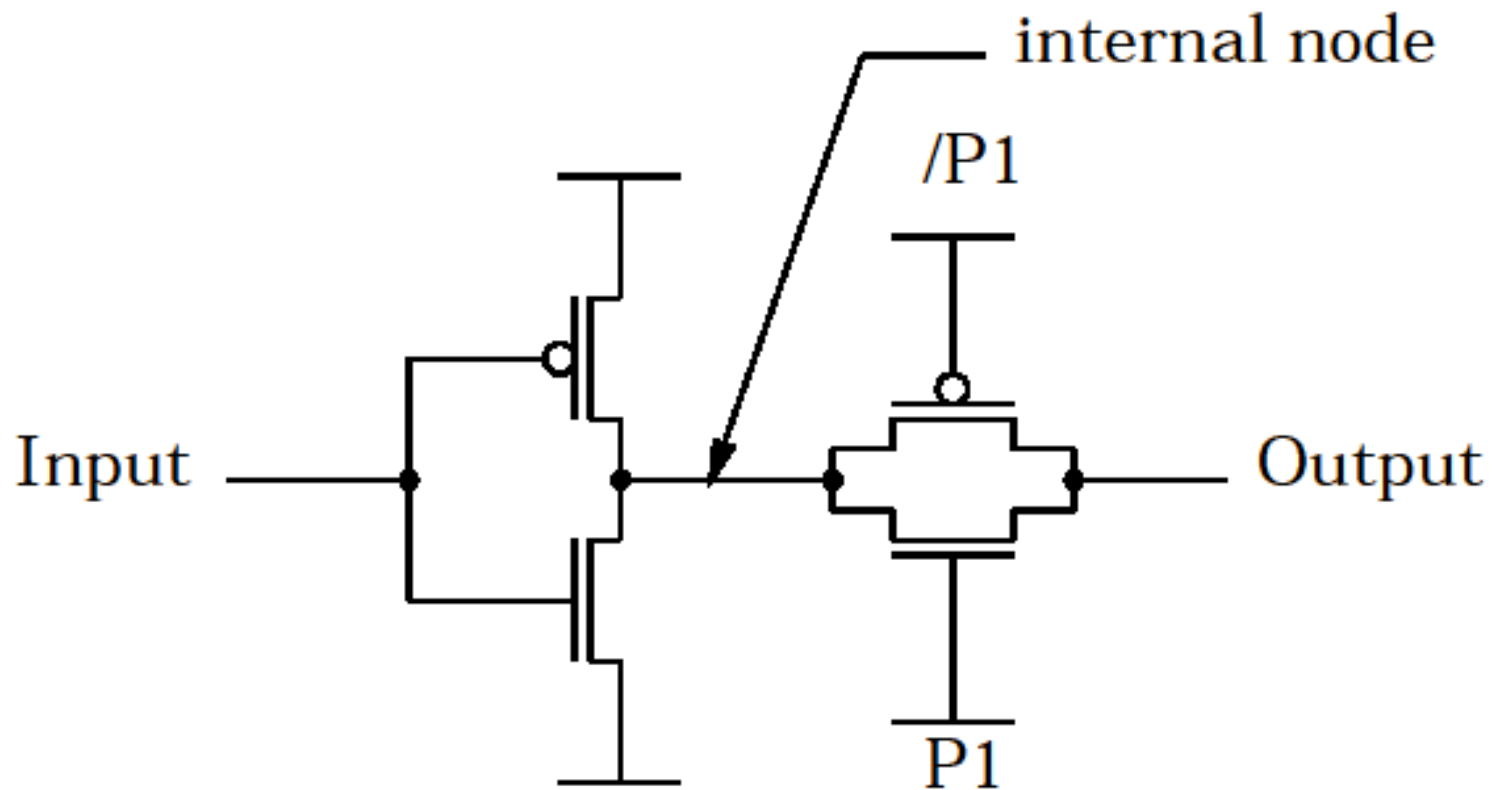
- Conventional Wisdom
 - Changing a gate's output consumes power
- More Detailed Understanding
 - Power consumed when a voltage drop occurs across a resistor
 - Capacitors store power that you can get back later
- Time-Power Trade-off
 - By accepting arbitrary circuit delay, can reduce power consumption to zero in asymptotic case.

Approach

- Rule
 - No transistor can turn on while there is a voltage drop across it
 - Eliminates resistive dissipation of heat
- Approach
 - Gates start out disconnected from their outputs, with power supplies at $V_{dd}/2$
 - Output starts at $V_{dd}/2$
 - Connect inputs to gates
 - Slowly connect gate to output
 - Slowly swing power supplies to V_{dd} and GND
 - Swing time determines both circuit speed and power
 - Once output sampled, disconnect gate from output, return power supplies to $V_{dd}/2$
 - All internal nodes reset to $V_{dd}/2$
 - Other circuitry responsible for resetting output to $V_{dd}/2$ before next input change

SCRL Inverter Example

- Can be extended to other logic gates using standard CMOS techniques



Pendulum: A Reversible Computer Architecture

- Complete Reversible Computer System
 - Reversible Operations
 - Reversible Logic
- Based on MIPS ISA
 - Simple implementation
 - Other researchers argue that CISC ISA may be better for reversibility
- Basic Design
 - Register file with forward and reverse ALUs
- Three Big Challenges
 - Memory/Register Access
 - Irreversible Operations
 - Control Flow

Memory/Register Access

- Traditional memory reads/writes destroy data
 - Clobber original contents of memory location or destination register
 - Remember that destruction of data is the fundamental consumer of power
- Solution: all memory/register accesses implemented as exchanges
 - LD/ST becomes exchange of one value with another
 - No information destruction
 - Pool of scratchpad locations provided for cases (register read) where no source for exchange exists
 - Preset to zero
 - Exchange may be distributed over multiple cycles

Irreversible Operations

- Many basic arithmetic operations are reversible
 - $A = A + B$ reversible unless overflow
 - Two-input instruction format makes more operations reversible
- Some aren't
 - $A = A$ or B not reversible
 - Fundamental destruction of information
- Solution: Garbage Stack
 - Whenever an irreversible operation is executed, push enough information onto the stack to allow reversibility
 - Pop information off of stack when running in reverse
- Alternate Approach
 - Replace irreversible operations with universal, reversible ones
 - Requires change to programming approach (compiler)

Control Flow

- Need to be able to reverse branches to run program backwards
- Problems
 - When to execute reverse branch?
 - Come-from statement
 - Branches only allowed to target instructions after come-from
 - Where does branch come from?
 - Multiple sources for one branch destination
 - Push PC onto garbage stack when execute branch