Energy-Aware Computing

Lecture 7: Micro-architecture level techniques

Classification

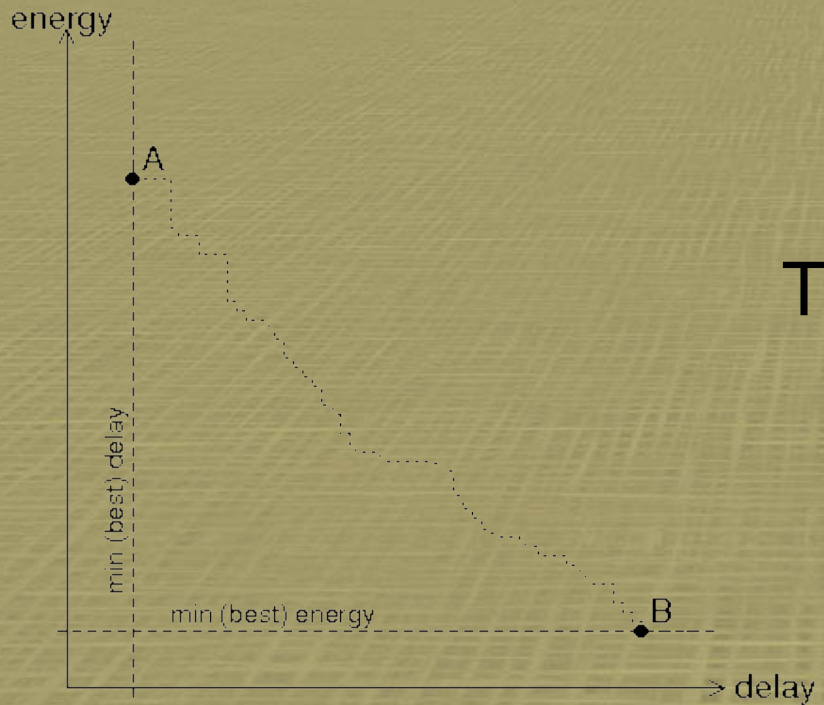
- Static
  - Fixed at design time
  - Win-win methods
    - Clever design improving power/energy without negative impact on speed
  - Win-loose (a little) methods
    - Trade off an acceptable speed drop for a large improvement in power/energy
- Dynamic
  - Adaptable at run-time. Can take advantage of win-loose-a-lot techniques

Categories - 2

- Design space variables:
  - Time
    - Not good for energy reduction on its own
  - Supply voltage
    - Architecture-driven voltage scaling
    - Dynamic voltage (and frequency) scaling
  - Architecture
    - Switched capacitance and IPC

- This lecture's focus is on static, mostly win-win methods

Dynamic methods



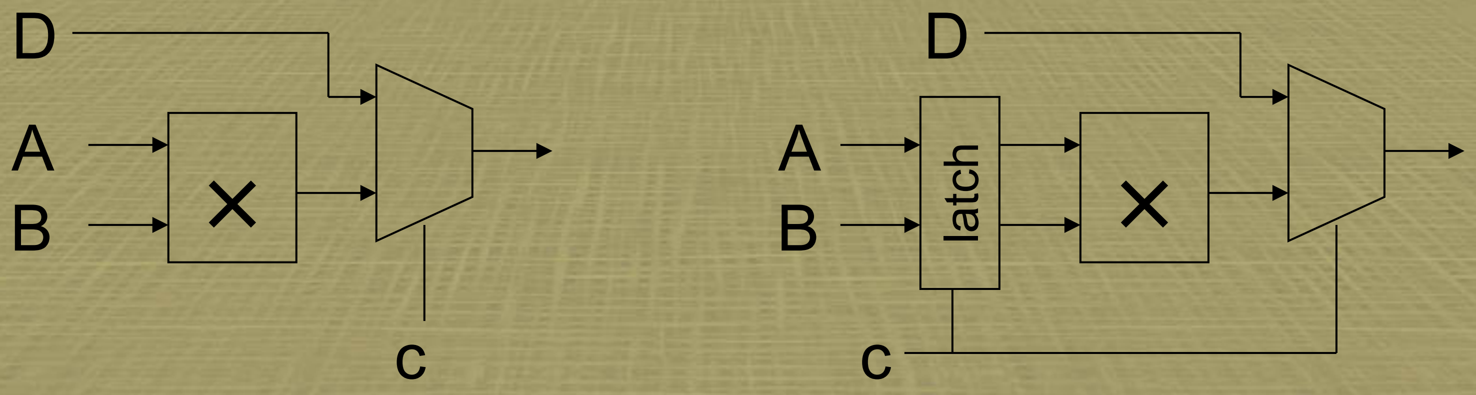
Trade-off time for energy

- Useful for systems with significant idle time
  - No change in perceived performance
- Useful when user wants to be able to make decisions (battery life vs quality of service)

Static method list

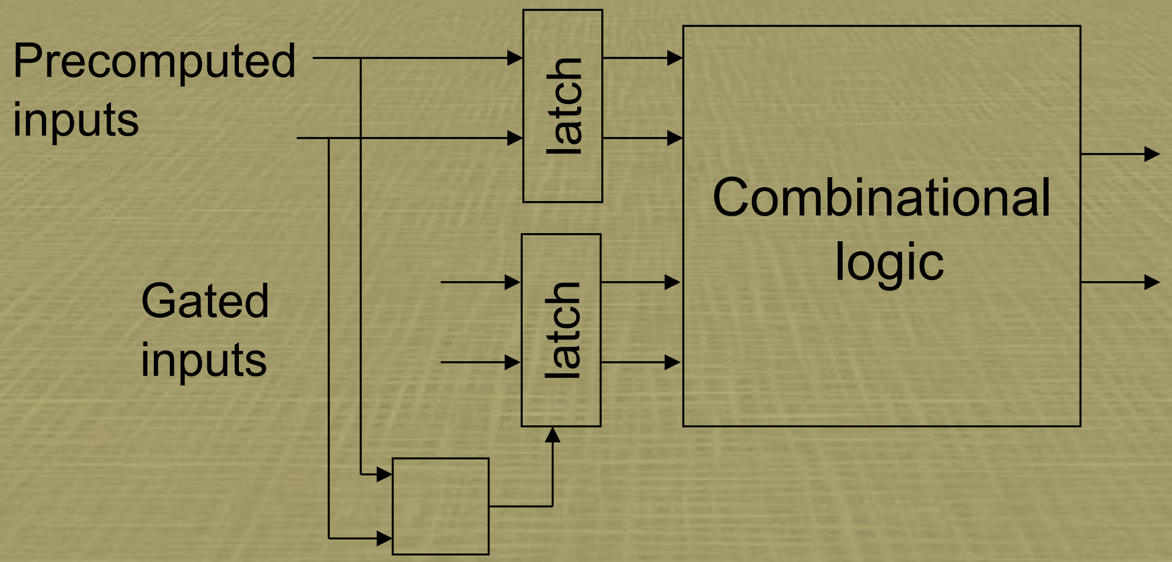
- Guarded evaluation
- Pre-computation
- Common case computation
- Signal gating
- Narrow-width operands
- Signal encoding

Guarded evaluation



- Reduce switching activity by blocking inputs when output is not needed
- Load of control signal is increased, so this is not always a winner

Pre-computation

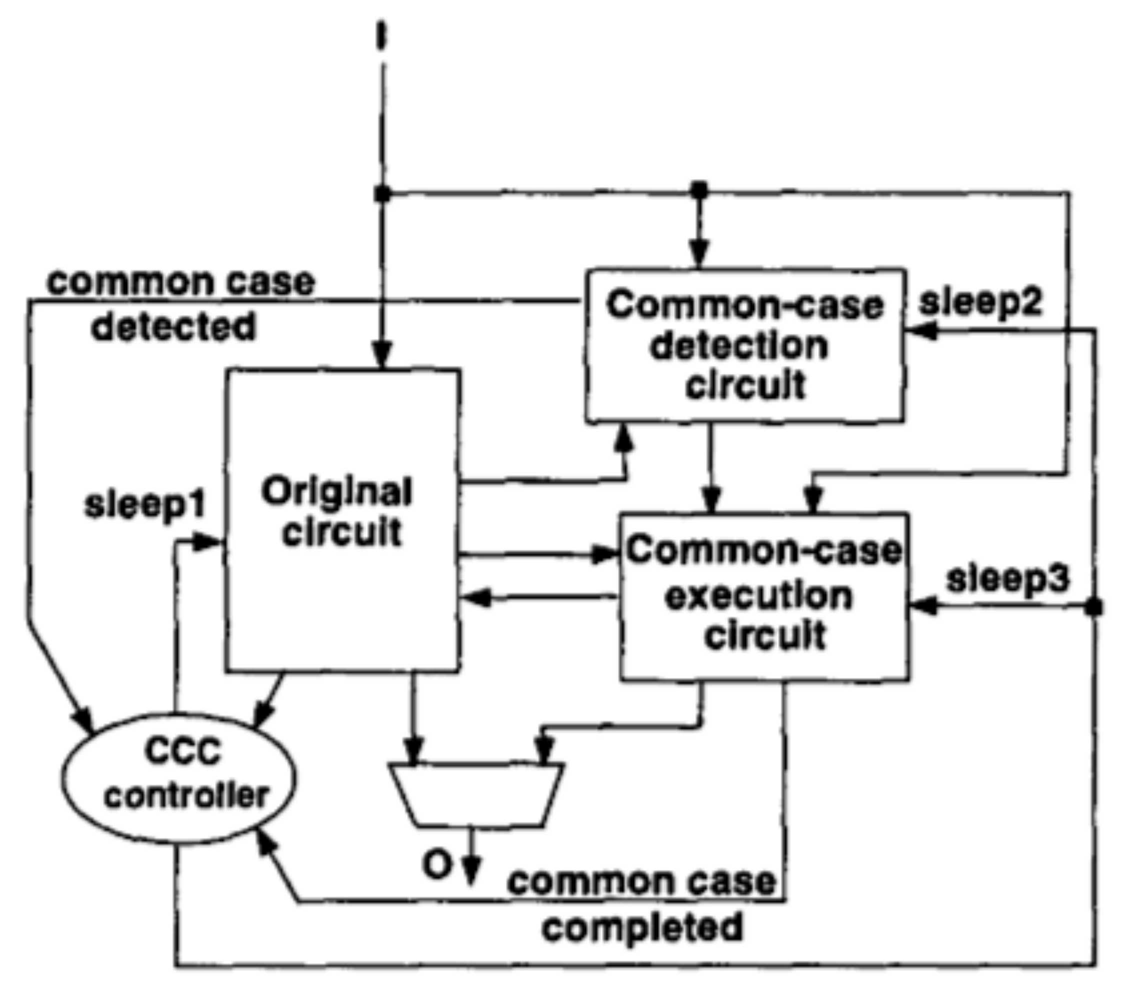


- Disable inputs which don’t affect the output
- Example: multi-bit comparator

Common case computation

- Some computations are more common than others
- Hardware units are typically designed for the general/broad case
- If common cases can be detected easily and computed fast, with low power consumption, …

Common case computation

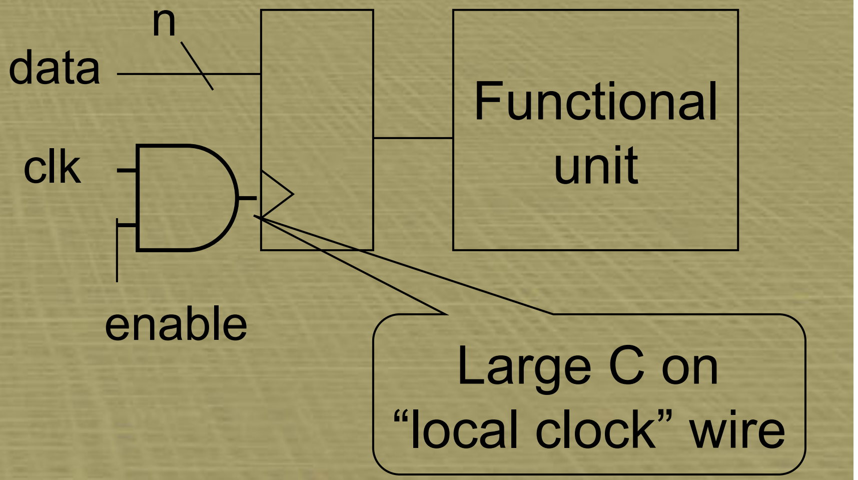


Signal gating

- Mask unwanted transitions on a high-load signal
- Using AND/OR gate, latch, etc.
- The most active signal is the clock
- Some early processors spent as much as 40% of their power budget on the clk
- Clock-gating is used in most systems

Clock gating

- Idle units don't need to be "clocked"
- Similarly for stalled pipe stages
- Combines guarded evaluation with reduction of activity on clock network



- If the functional unit is implemented using dynamic logic, savings are even greater

Clock gating problems

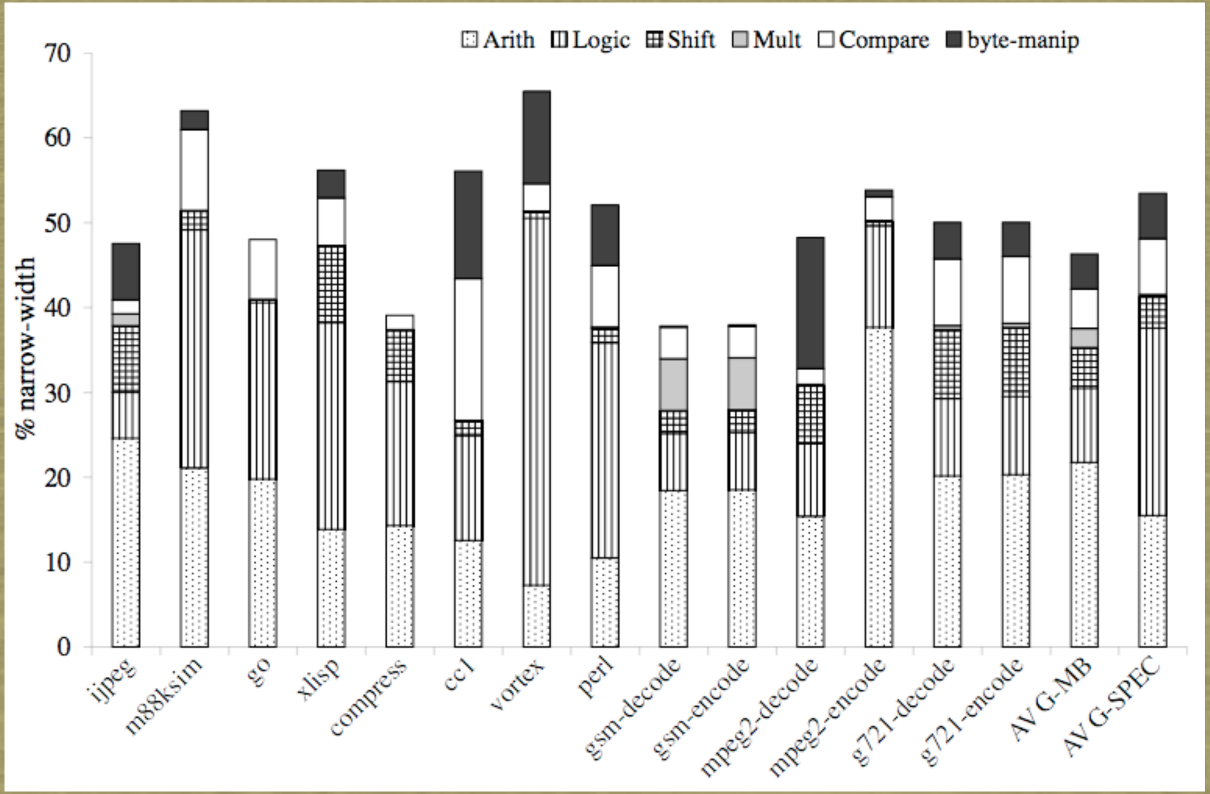
- Enable signal is usually on critical path
- Must avoid causing *clock glitches*
  - Enable signal's transitions must be controlled
- Delay on clock signal
  - Clock delay must be matched across the whole chip. If some parts use AND gates and others buffers this matching could be difficult
- Supply voltage drop due to high di/dt

Narrow-width operands

- Most modern processors operate on 64-bit words
  - Adders, shifters, etc. all 64-bit wide
- Many operations use much less than the full width
  - E.g. in address calculation the offset is small, around 16bits
- Idea: can the actual width be detected and the unused parts of the functional units gated off?

Operand width statistics

Percentage of instructions with both operands below 16 bits. From Brooks, Martonosi ACM TCS’00



Implementation #1

- By Brooks, Martonosi TCS’00
- Assume 64b word
- Split operands into 16 LSb and 48 MSb
- Detect at the ALU output when 48MSb are all 0 or 1 and use an extra tag bit to record this info
- Each operand’s tag is used as an enable signal for the input register holding the upper 48 bits of the operand
- 55-58% reduction of integer FUs power

Implementation #2

- Significance compression Canal et al micro'00
- 3 bit tag for 32 bit words
  - Each tag bit corresponds to each of the MS 3 bytes: 1 if sign extension, 0 otherwise
- Various implementations possible
  - Byte serial: datapath is 1 byte wide
  - Skewed: operation spread over a number of pipeline stages
  - Fully parallel: similar to implementation #1

Signal encoding

- Signal encoding directly affects switching activity
- E.g. sign-magnitute to be better for DSP applications (Chandrakassan 95)
  - Most registers/values are close to 0
  - Changing from positive to negative causes all MSBs to switch frequently
- But hard to design functional units for non-standard encodings
- Much easier to use “low-power” encodings in busses
  - Off-chip busses have much higher C

Address busses

- Address busses have regular sequential or stride behaviour
- *Gray code* minimises sequential transitions
- Bin code: 000, 001, 010, 011, 100, 101, 110, 111
- Gray: 000, 001, 011, 010, 110, 111, 101, 100

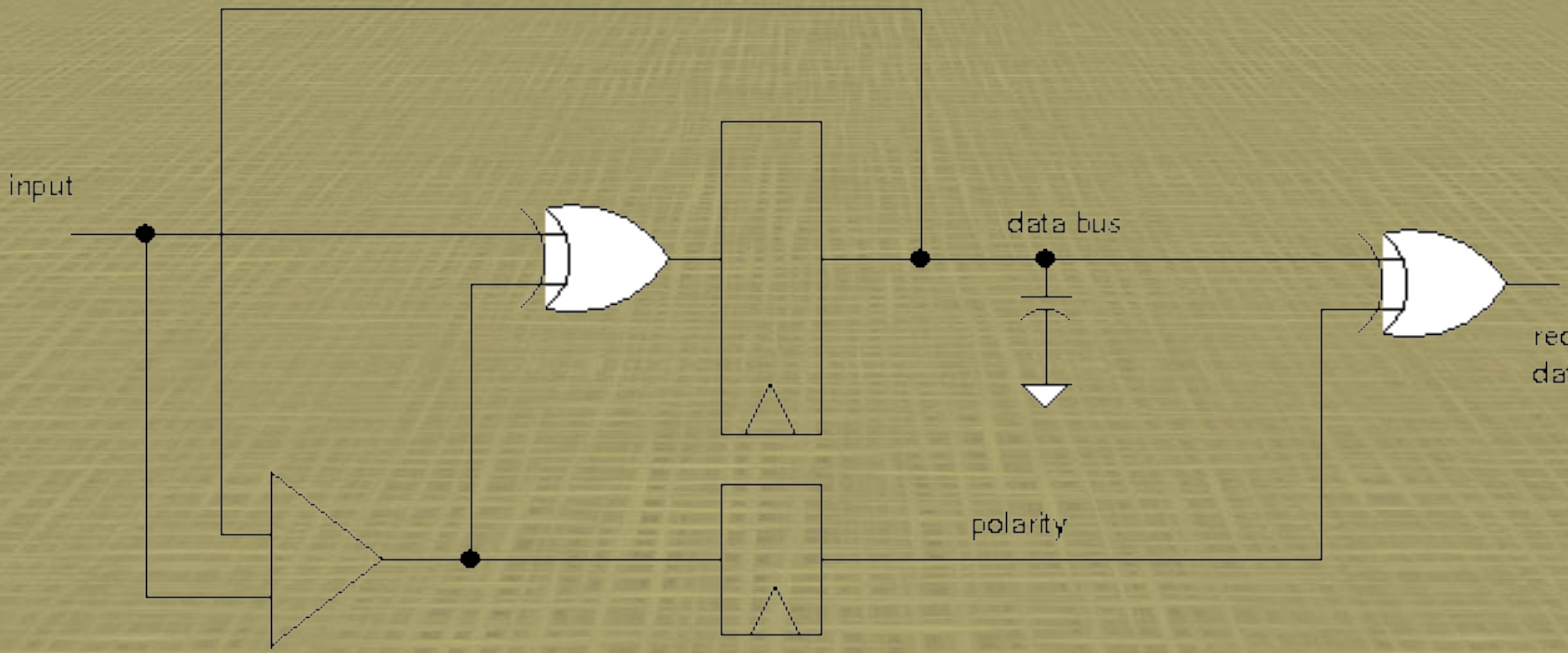
T0 code

- Benini et al GLVLSI’97
  - Add a signal named INC
  - Freeze the address lines for consecutive addresses and assert INC
  - Add a counter at the destination
- Better than Gray when probability of having consecutive addresses is > 0.5
  - No transitions at all for consecutive addresses!
- Easy to know when next address is consecutive
  - Branch predictor output
  - Some ARMs used this to speed up memory access

Bus invert encoding

- Stan, Burleson, TVLSI 3(1) 1995
  - Decide if sending the true or complimentary signal leads to fewer toggles
  - Add polarity signal to the bus
- Used for both address and data busses
- Lots of variations published since

Bus invert implementation



Bus invert encoding

- Maximum of n/2 bits can toggle
  - Could be n without encoding
- Additional logic has area, power, delay overhead
- For uniform random signals up to 25% of toggle reduction

Summary

- Categories of micro-arch methods
  - Static, dynamic, …
- Low-level static methods
  - Guarded evaluation
  - Pre-computation
  - Common case computation
  - Clock gating
  - Operand width reduction
  - Bus encoding