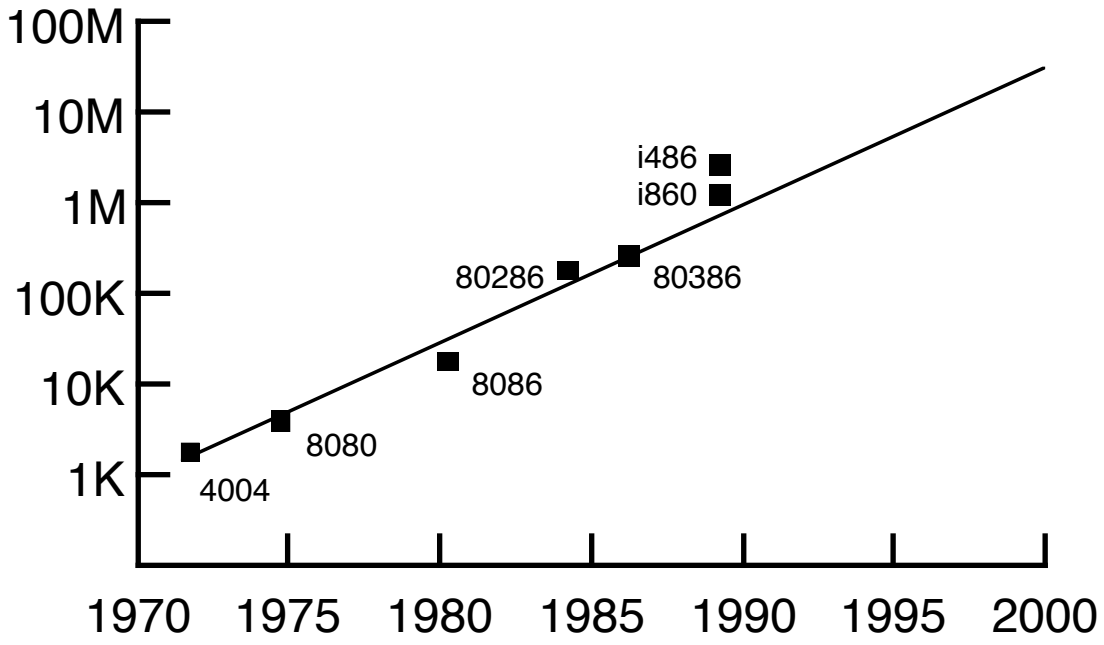
Computer design

Modularity, control, timing

P.J. Drongowski
SandSoftwareSound.net

Growth of complexity



Processor	Transistors
4004	2,300
8080	6,000
8086	29,000
80286	134,000
80386	275,000
i860	1,000,000
i486	1,200,00

Microcomputer Solutions, Intel Corporation, 1989

Productivity

- Software
  - One line of code per hour
  - Designed, debugged, integrated, documented
- Hardware
  - 100,000 transistor design
  - 30 to 40 person-years of effort
  - Roughly 1.5 transistors per hour
- Complexity
  - Software: 100,000 to 1,000,000+ lines of code
  - Hardware: One million transistors

Implications

- Beyond intellectual capability of any single engineer
- Development must be a team effort
- Interpersonal communication is essential
  - Difficult to agree on meaning of common terms
  - "Massive semantic by-pass"
- Better tools and techniques needed
  - Increase productivity
  - Assure correctness and performance
  - Support product throughout lifecycle

Modular design

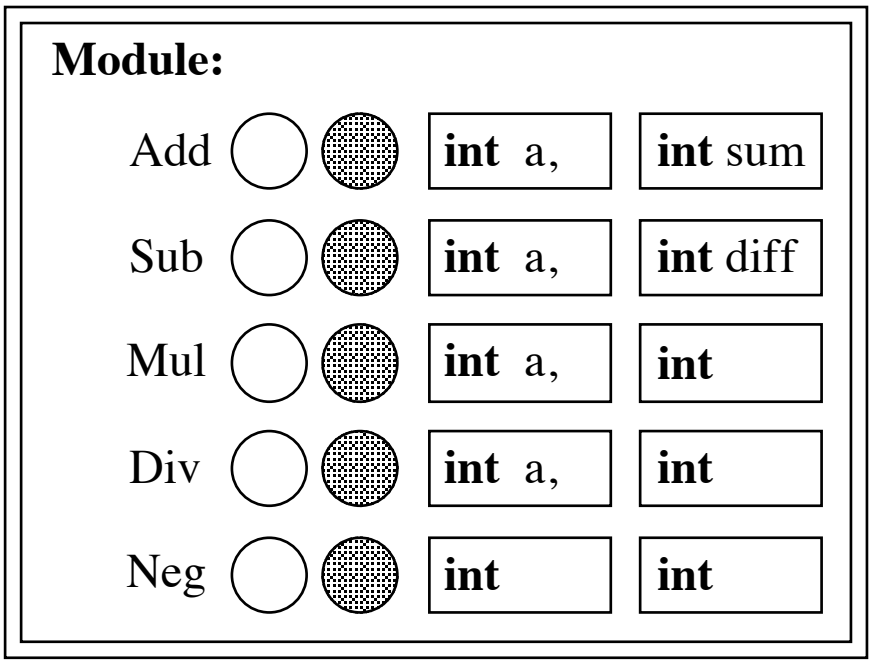
- Methodology
  - Partition design into subunits
  - Design, code, test, debug subunits in isolation
  - Integrate subunits into (sub)system and test

- Advantages
  - Reduces problem compexity
  - Permits team implementation effort

- Problems and needs
  - Good interface specifications
    - Build to specification
    - Avoid interface errors during integration
    - Test to specification
  - Good communication
    - Make assumptions explicit
    - Semantic agreement
  - Isolation
    - Transparency
    - Hide design decisions (algorithms, data structures)
    - Enhances maintainability
  - Yo-yo design
    - Capture functionality (top-down)
    - Assure performance (bottom-up)
    - Need "crystal ball" during top-down design
    - Poor choice means expensive fix later

Modules

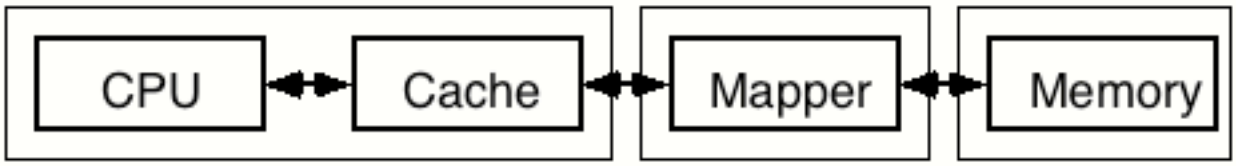
- A module is a black box
- The contents of the black box are unknown
- A module implements one or more operations
  - Button to invoke an operation
  - Input slot(s) to send arguments
  - Output slot(s) to receive results
  - Completion signal (e.g., indicator light)
- Invocation procedure ("Coke machine" model)
  - User sends arguments to input slots
  - User pushes button
  - Module commences operation
  - When finished, completion is signalled
  - User removes results from output slots



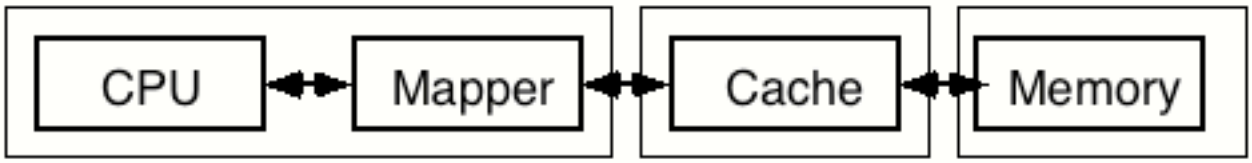
Partitioning design decisions

- Top-down design
  - Management of system complexity
  - Divide and conquer system behavior
  - Partitioning has performance implications
  - What if we choose the wrong system structure?
- Bottom-up design
  - Cannot easily manage behavioral complexity
  - Speed, space, power, etc. can be accurately estimated
- "Yo-yo" design
  - Practical approach
  - Partition at top then estimate from the bottom
  - Iteratively evaluate/modify the partition

Example: Placing a cache memory (physical partition)



- Cache virtual addresses
- Fast access to program data
- Avoid mapping delay



- Cache physical addresses
- Slower access to cache data

Software modules

- Access controlled subunits
  - Visible interface
    - Exported procedures that may be called
    - Exported variables and data structures
  - Implementation (transparent operation)
    - Hidden functions
    - Hidden data structures
  - Degrees of access control
    - Public export (anyone can call)
      - Make name global
      - Import name through `extern`
    - Directed export (calling modules explicit)
      - Explicitly identify operations and legal users
      - Caller imports operation
- Separately coded and compiled

Communications conventions

- Modularity works through conventions
- Compiler enforces / implements convention
- Subroutine call
  - Save working registers
  - Put arguments on stack (or in display)
  - Call and save return address
  - Result on stack or in general register
  - Restore working register values
  - Examples: C or C++
- Message passing
  - Receiving object has a dictionary of methods
  - Send message with method name and arguments
  - Receiver invokes method
  - May reply to sender with result message
  - Examples: Smalltalk or Actors

C++ modularity

- Object-oriented C dialect
- Classes define "types" of objects

```
class class-name {
    private-variables
public:
    public-interface
} ;

void
class-name::member-function()
    {
    ...
```

- Objects are instantiated through declaration

```
class-name instance-name
```

- Apply member function to object instance

```
instance-name.member-function( ... )
```

- Constructor
  - Initialization
  - Member function with same name as class
  - Storage allocation through **new**
- Destructor
  - Clean-up after use
  - Function name is ~ *class-name*
  - Storage deallocation through **delete**
- Inheritance

C++ inheritance example

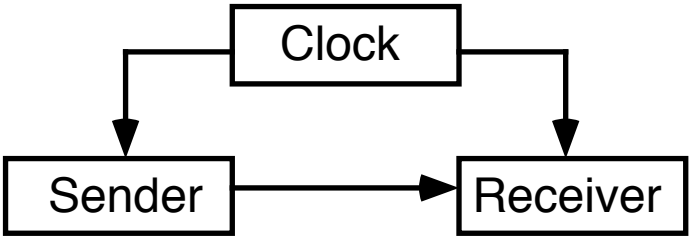
```
class Register {
  unsigned Value ;
public:
  void Clear() { Value = 0 ; }
  void Load(int NewValue) { Value =
NewValue; }
  unsigned Read() { return( Value ) ; }
  Register() { Value = 0 ; }
} ;

class Counter : public Register {
public:
  void Incr() { Load(Read() + 1) ; }
  void Decr() { Load(Read() - 1) ; }
} ;

class ShiftRegister : public Register {
public:
  void Left()  { Load(Read() << 1) ; }
  void Right() { Load(Read() >> 1) ; }
} ;
```

Hardware modules

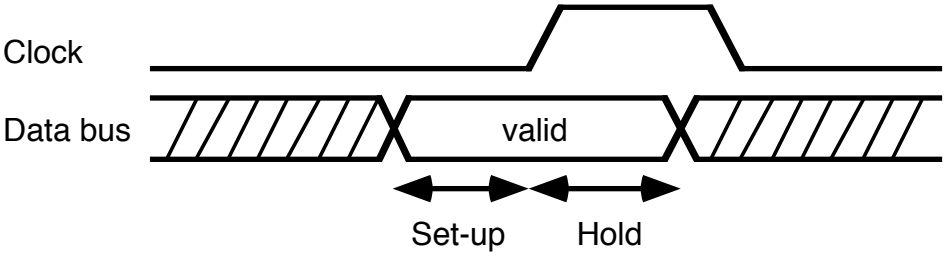
- Monolithic block of circuitry
- Signal ports (physical connection points)
  - Signalling protocol
  - Data representation (unisigned, int, etc.)
  - Timing (assumptions / constraints)
  - Serial versus parallel (low pin-out versus speed)
  - Correct electronic operation
    - Inputs put load on user outputs
    - Outputs must drive user result inputs
- Consistently apply discipline as in software design
- Asynchronous operation
  - Similar to "Coke machine" model
  - Supply arguments, make request
  - Produce results, signal completion
  - Computation can take as long as it needs
- Synchronous operation
  - Sender and receiver both step to common clock
  - Agree to exchange data within time window
  - Produce results within prearranged period
- Communication "styles"
  - Single - phase synchronous
  - Two phase, non-overlapping clock
  - Four cycle handshaking
  - Two cycle handshaking

Synchronous design

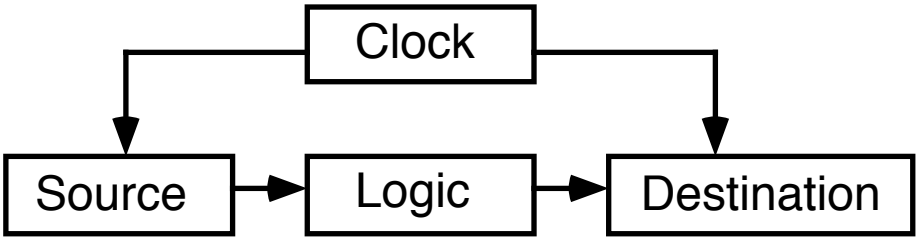
- Simple sender-receiver communication



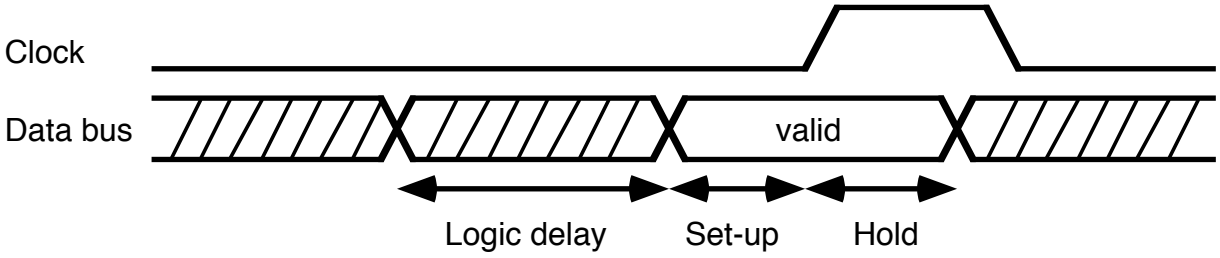
- Simplified timing



- Standard synchronous model

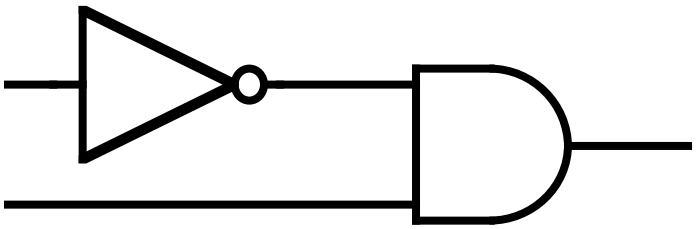


- Synchronous timing constraints



Synchronous time constraints

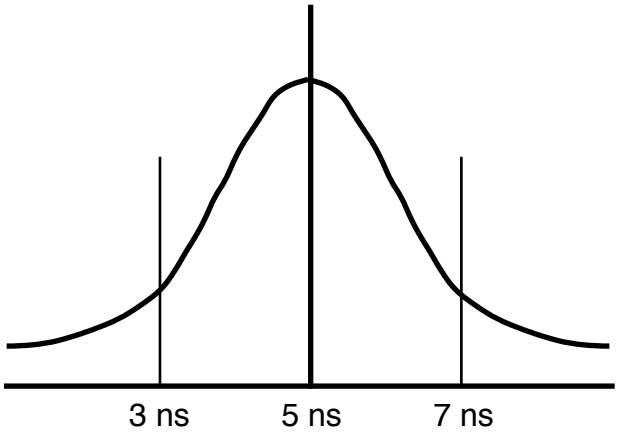
- Clock period is the sum of:
  - Compute time
  - Set-up time
  - Hold time (or pulse width if longer.)
- Estimate path delay by summing gate delays.



- Data book delay specification.

Minimum	Typical	Maximum
3 nsec	5 nsec	7 nsec

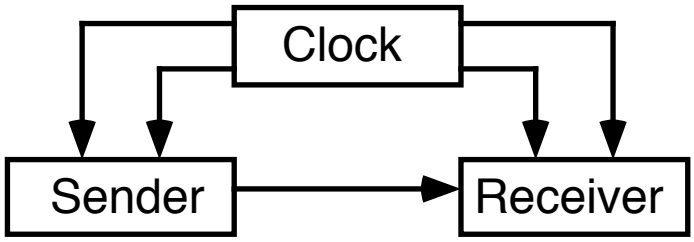
- Delay varies due to differences in manufacturing



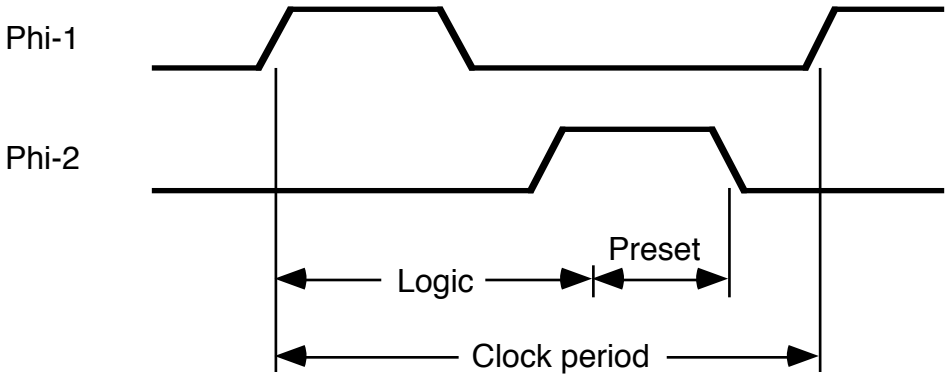
- A mix of slow parts will violate timing constraint.
- Defensive design and manufacturing.
  - Design in a safety margin.
  - Screen components for speed before assembly.
  - Use of worst case is overly conservative.
  - Tune clock after assembly (bad approach.)

Two phase, non-overlapping clock

- Similar to single phase synchronous style



- Timing

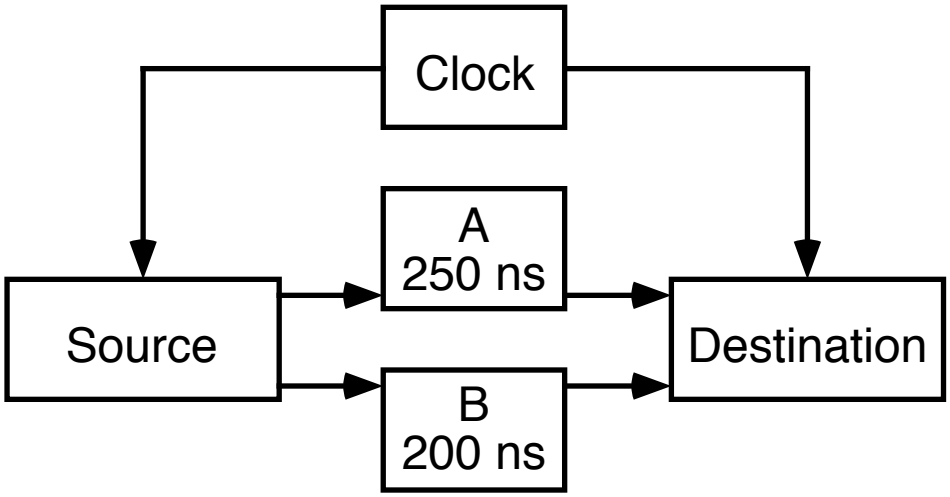


- Operation
  - Two clock phases: Phi-1 and Phi-2
    - When Phi-1 is high, Phi-2 is low
    - When Phi-2 is high, Phi-1 is low
    - Phi-1 and Phi-2 are *never* high at the same time
  - Two intermediate periods are needed when both are low
    - First period can accomodate logic delay
    - Second period is idle; make as short as possible
  - Compute during Phi-1; Store during Phi-2

- Control of charge flow
  - Analogous to canal locks (water ⇔ electric charge)
    - Open gate to allow charge into combinational logic
    - Close gate and compute
    - Open gate to release and store results
    - Close gate to put system in consistent idle state

Synchronous limitations

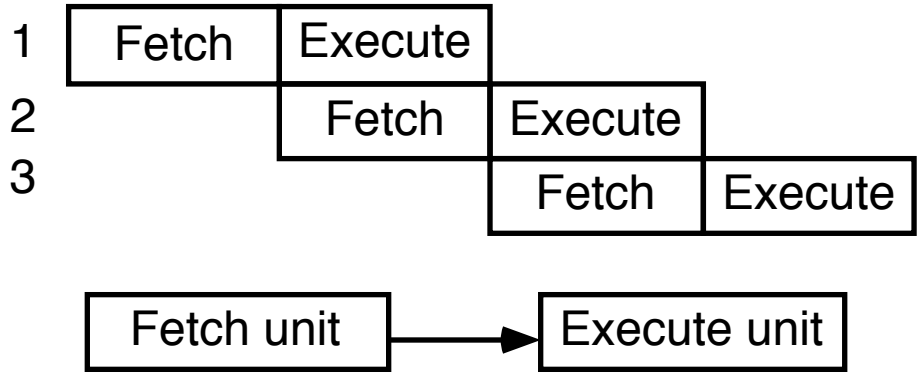
- Speed is limited by slowest component
- Min clock period is determined by max delay
- Max delay path is called "critical delay path"



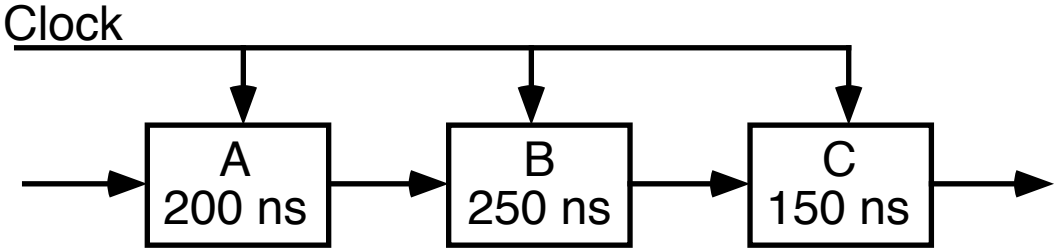
- Try to make unit B faster
  - Use a better algorithm or faster components
  - Example: Ripple carry adder versus carry-lookahead
- Unit A is over-designed
  - Slow unit A down
  - Try to use fewer components or lower power
- Use different clocking scheme
  - Poly-phase clock
  - Programmable clock periods
- Self-timed systems
  - Allow each unit to execute at its own speed
  - Synchronize when necessary

Synchronous pipeline

- Overlap computations in stages
- Example: Instruction lookahead



- Synchronous 3-stage pipeline (below.)
  - A, B and C operate in lockstep
  - Speed is determined by slowest unit (B)

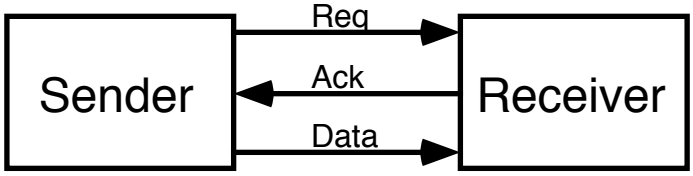


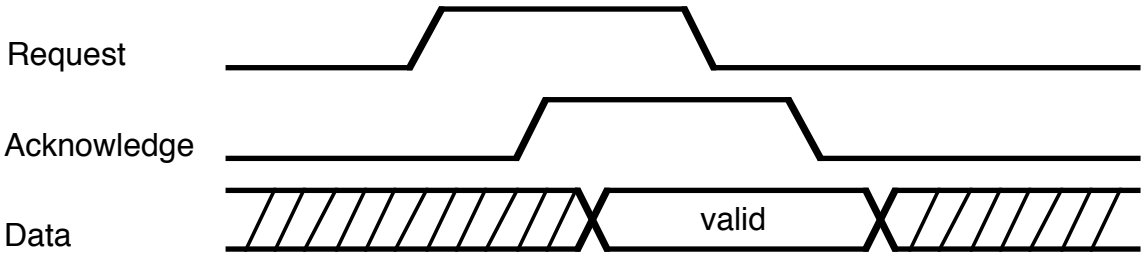
Pipeline problems

- Hazards
  - Data dependencies
  - Control dependencies
  - Collisions (resource conflict)
- Data dependency
  - Problem: Result is needed before it is stored
  - Example: Berkeley RISC
  - Solution: Detect hazard and forward result
- Control dependency
  - Problem: Branch occurs after prefetch
  - Example: Berkeley RISC and SPARC
  - Solutions
    - Flush and refill pipeline
    - Fill pipeline with no-op instructions
    - Delay the effect of the branch
      - ◊ Execute instructions already in pipe
      - ◊ Branch late
      - ◊ Compile code to use "extra" instruction effectively
      - ◊ Can successfully find work in 90% of the cases
- Collisions (resource conflicts)
  - Two instructions need the same resource
  - Detection and resolution
    - Static
      - ◊ Usually detect at decode stage
      - ◊ Conservative approach - hold until ready
    - Dynamic
      - ◊ Let instruction proceed
      - ◊ Detect and resolve at point of conflict
  - Usage counters
  - Scoreboarding

Four-cycle handshake

- Communication elements are on closed loop path
- No request until completion of previous operation
- Permit imposition of arbitrary delay
- Delay insensitive, self-timed signalling convention
- Operation
  - Sender asserts request (to send) line
  - Receiver asserts acknowledge (ready to receive)
  - Sender sets-up data on bus
  - Sender drops request, indicating data ready (valid)
  - Receiver drops acknowledge after data capture
- Disadvantages
  - Slow speed due to two way signalling
  - Two-cycle version eliminates some overhead



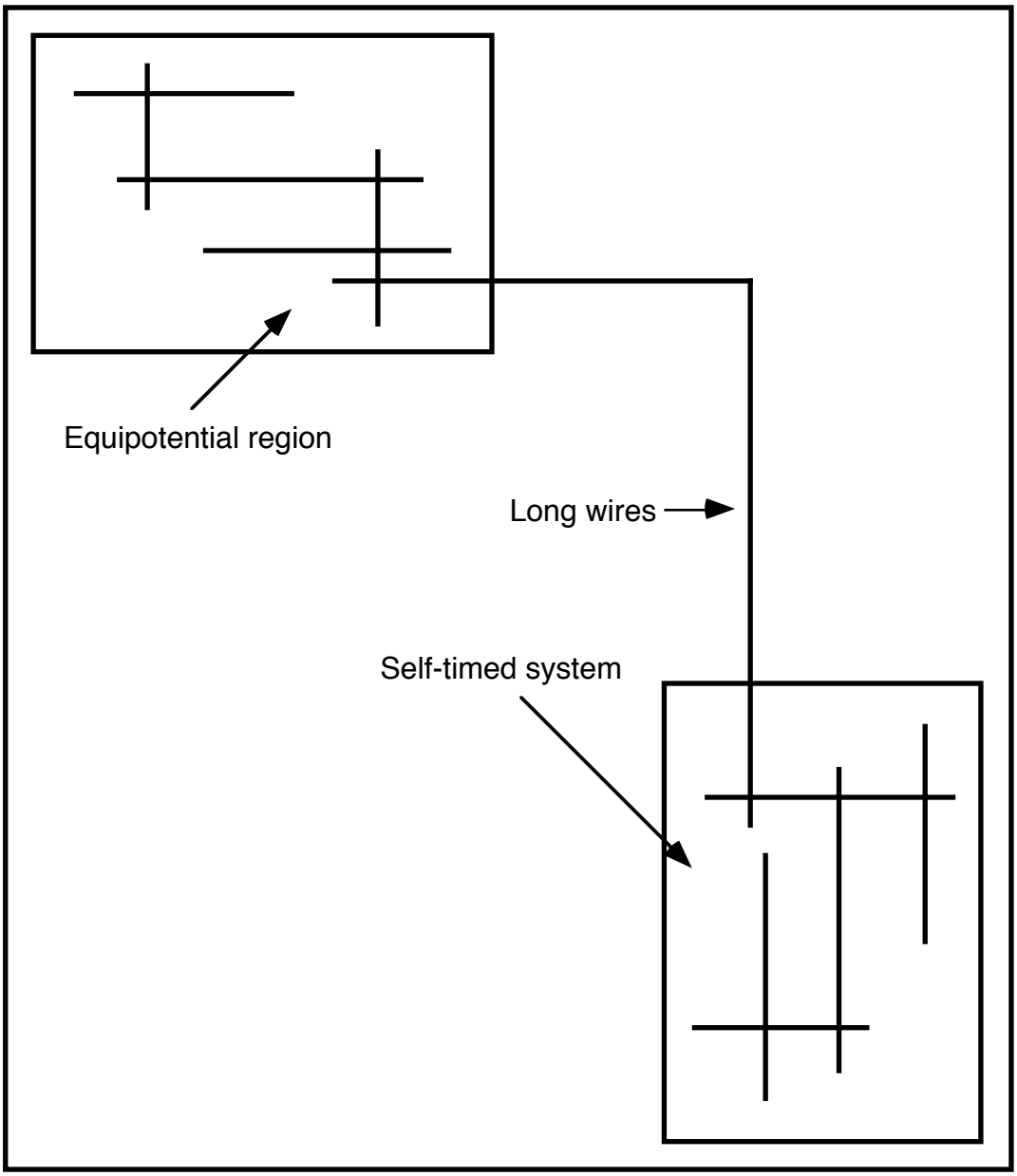


Self-timed systems

- Sources of delay
  - Switching time
  - Wire capacitance
  - Off-chip connections
  - Diffusion delay (quadratic in length)
- Effect of delay
  - Long compute times
  - Clock skew
- Equipotential regions (Seitz)
  - Equalization time is relatively short
  - Synchronous timing constraints are satisfied
- How big should a region be?
- Synchronization failure
  - Synchronous sampling is risky
  - Finite probability of failure
- Use asynchronous signalling between regions
- "Micropipelines," Ivan E. Sutherland, CACM, Volume 32, Number 6, June 1989, pg. 720-738. (Turing Award paper.)
  - Request/acknowledge interlock between pipe stages
  - Use delays plus Muller C-element

Self-timed organization

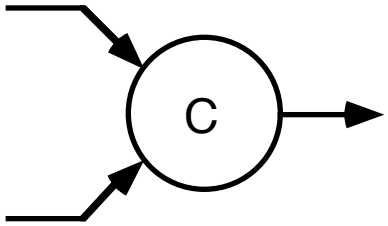
- Let each "time zone" have its own clock
  - Signals will equalize in same short period of time
  - Exploits physical locality
- Communication between self-timed systems
  - Systems must synchronize to communicate
  - Use four-phase signalling or synchronizers
  - Try to communicate infrequently



Event logic

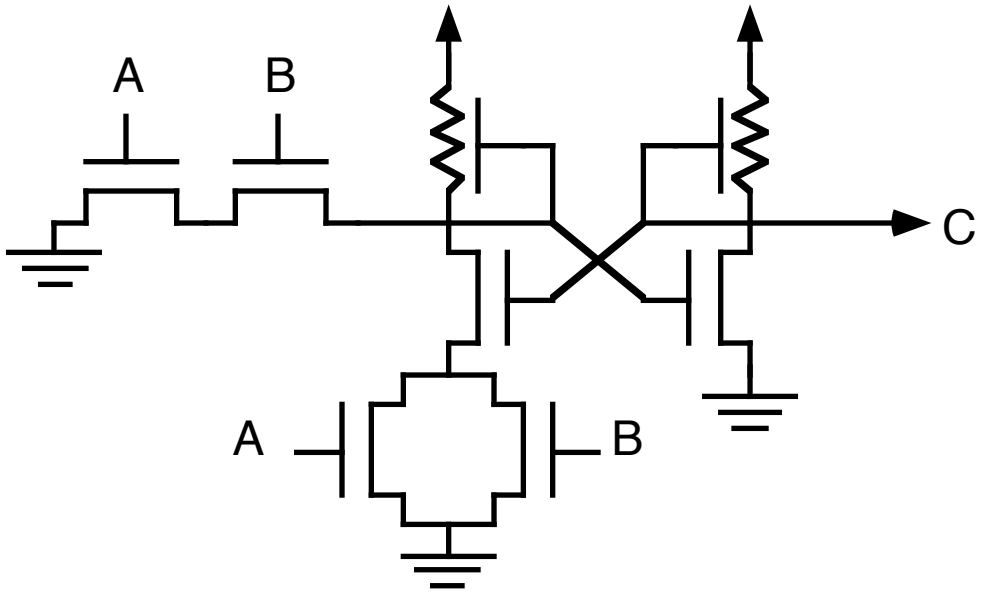
- Use signal transitions for control
- Example: Request-acknowledge handshake
- AND-merge (join, rendezvous.)
  - Muller C-element
  - If inputs match, copy their state to output
  - If inputs differ, hold previous state
- OR-merge
  - Exclusive-OR gate
  - If an input changes, then change output
- Toggle
  - Steer events alternatively to its outputs
  - First input can be set at start-up
- Select
  - Steer events according to Boolean input
- Call
  - Procedure invoked by either of two clients
  - Remember identity of caller
  - Return "done" after completion
- Arbiter
  - Two clients request shared resource
  - Grants mutually exclusive access
  - Must wait for completion

Muller C-element

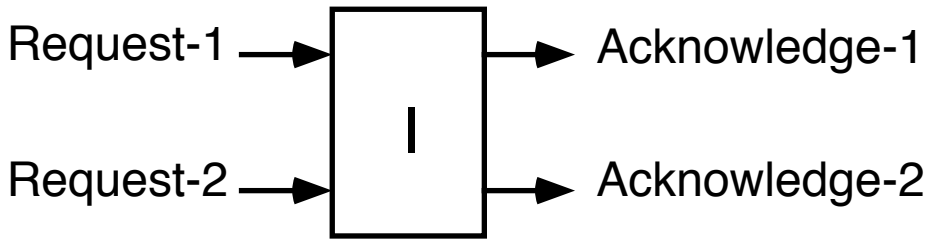


Logic symbol

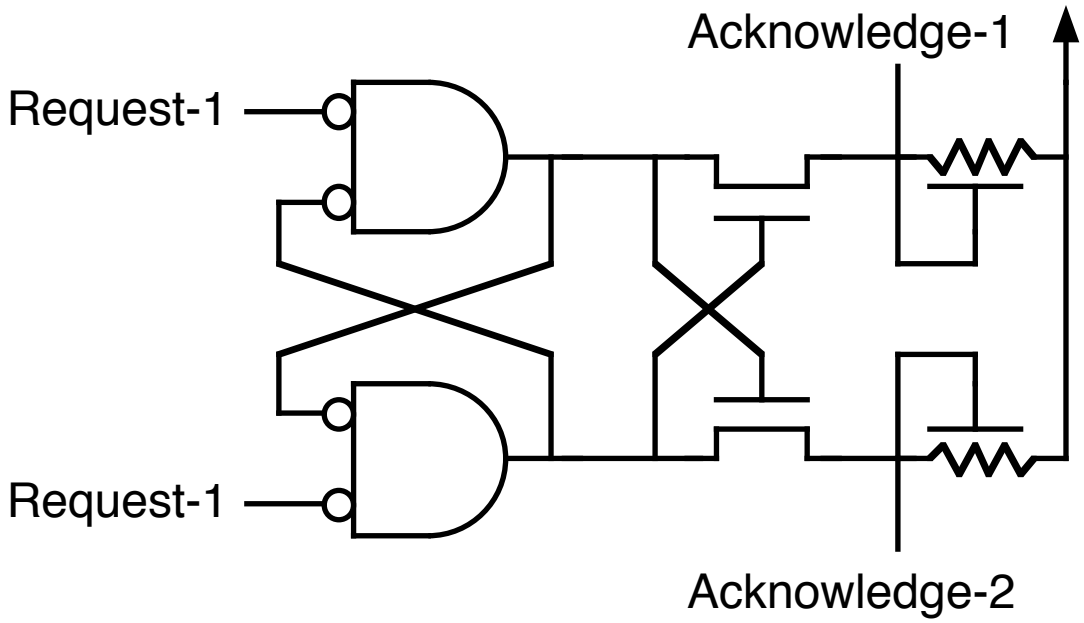
nMOS implementation



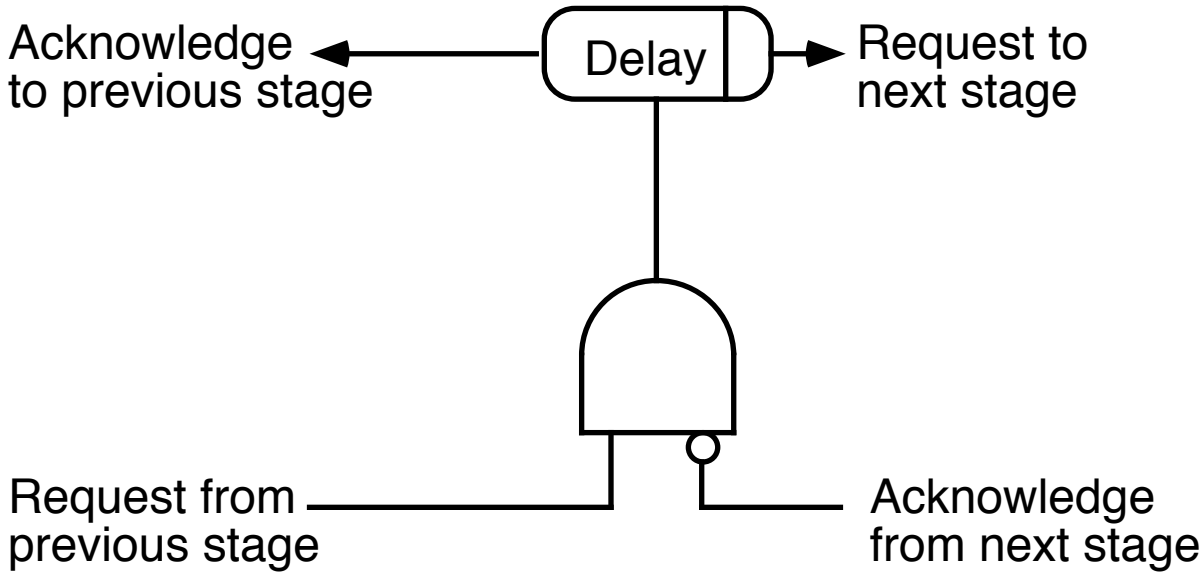
Interlock element



nMOS implementation



Micropipeline control circuit



- Stage state rule

  If *predecessor and successor differ in state*
  then *copy predecessor's state*
  else *hold present state*

- Each loop contains one inverter and will alternate
- Composability
  - All stages share the same request/acknowledge signals
  - Stages are easily composed into longer pipes
- Scheme uses two phase signalling

Synchronization failure

- "System timing," Charles Seitz, Chapter 7 in "Introduction to VLSI Systems," Carver Mead and Lynn Conway, Addison-Wesley, 1980

- Philosophical problem
  - Jean Buridan, French philosopher
  - Case of the hungry dog
  - Dog is equidistant from two equal amounts of food
  - Equally attracted to each bowl of food
  - Dog will starve (paradox)
  - State of equilibrium

- Metastable condition
  - Unstable equilibrium
  - Condition may persist indefinitely
  - Behavior in cross-coupled circuits
    - Output voltage in range around logic threshold
    - Cannot reliably interpret as high or low

- Synchronous system reading asynchronous signal
  - Signal changes are *not* discrete
  - Unequal delays (below) will cause illegal state
  - Solutions
    - Chain together synchronizers
    - Stoppable clock (synch stop, asynch start)