Review of Digital Logic

COMS W4995-02

Prof. Stephen A. Edwards
Fall 2002
Columbia University
Department of Computer Science

Synchronous Digital Logic Systems

Raw materials: CMOS transistors and wires on ICs

Wires are excellent conveyors of voltage

- Minimal leakage
- Fast, but not instantaneous propagation
- Many orders of magnitude more conductive than glass

CMOS transistors are reasonable switches

- Finite, mostly-predictable switching times
- Nonlinear transfer characteristics
- Voltage gain is in the 100s

Philosophy

Must deal with unpredictable voltages and unpredictable delays

Digital: discretize values to avoid voltage noise

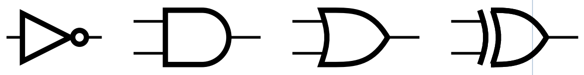
- Only use two values
- Voltages near these two are “snapped” to remove noise

Synchronous: discretize time to avoid “time noise”

- Use a global, periodic clock
- Values generated before the clock are ignored until the clock arrives

Combinational Logic

Boolean Logic Gates



Inverter

A	Y
0	1
1	0

AND

AB	Y
00	0
01	0
10	0
11	1

OR

AB	Y
00	0
01	1
10	1
11	1

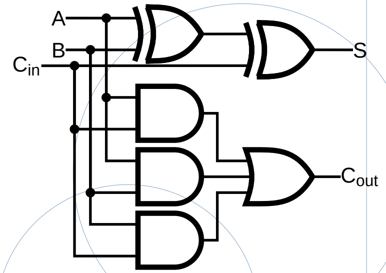
XOR

AB	Y
00	0
01	1
10	1
11	0

A Full Adder

Typical example of building a more complex function

ABC_{in}	C_{out}	S
000	0	0
001	0	1
010	0	1
011	1	0
100	0	1
101	1	0
110	1	0
111	1	1

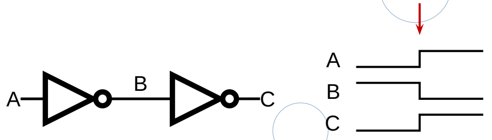


Most Basic Computational Model

Every gate is continuously looking at its inputs and instantaneously setting its outputs accordingly

Values are communicated instantly from gate outputs to inputs

All three switch at exactly the same time

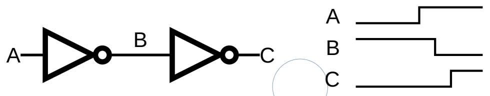


Simple Delays

Real implementations are not quite so perfect

Computation actually takes some time

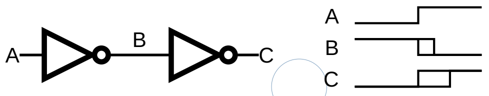
Communication actually takes some time



Delay Ranges

Delays are often partially unpredictable

Usually modeled with a minimum and maximum



Busses

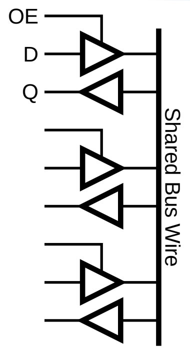
Wires sometimes used as shared communication medium

Think “party-line telephone”

Bus drivers may elect to set the value on a wire or let some other driver set that value

Electrically disastrous if two drivers “fight” over the value on the bus

Implementing Busses



Basic trick is to use a “tri-state” driver than can choose to not drive a wire.

Data input and output enable

When driver wants to send values on the bus, OE = 1 and D contains the data

When driver wants to listen and let some other driver set the value, OE = 0 and Q returns the value

Four-Valued Simulation

Wires in digital logic often modeled with four values: 0, 1, X, Z

X represents an unknown state

- State of a latch or flip-flop when circuit powers up
- Result of two gates trying to drive wire to 0 and 1 simultaneously
- Output of flip-flop when setup or hold time violated
- Output of a gate reading an “X” or “Z”

Z represents an undriven state: Value on a shared bus when no driver is output-enabled

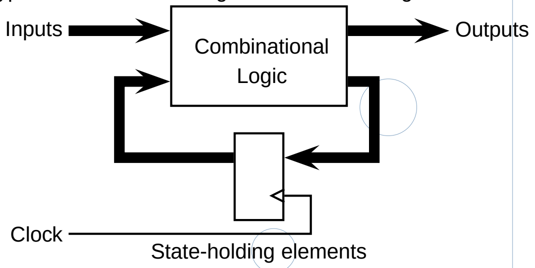
Sequential Logic and Timing

Sequential Logic

Simply computing functions usually not enough

Want more time-varying behavior

Typical: combinational logic with state-holding elements



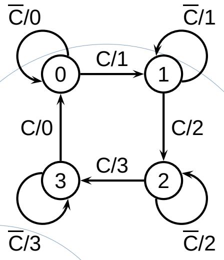
State Machines

Common use of state-holding elements

Idea: machine may go to a new state in each cycle

Output and next state dependent on present state

E.g., a four-counter

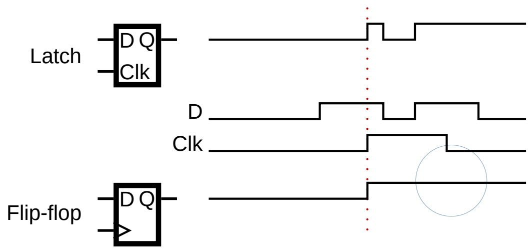


State-holders: Latches & Flip-Flops

Latch	**Flip-flop**
Level-sensitive	Edge-sensitive
Transparent when clock is high	Always holds value
Holds last value when clock is low	New value sampled when clock transitions from 0 to 1
Cheap to implement	More costly to implement
Somewhat unwieldy to design with	Much easier to design with

Latches & Flip-Flops

Timing diagrams for the two common types:

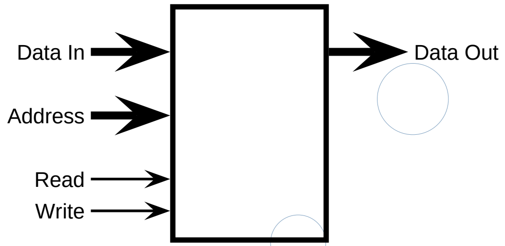


RAMs

Another type of state-holding element

Addressable memory

Good for storing data like a von Neumann program



RAMs

Write cycle

Present Address, data to be written

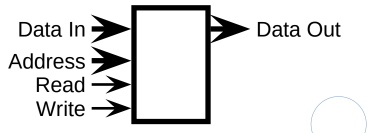
Raise and lower write input

Read cycle

Present Address

Raise read

Contents of address appears on data out



Setup & Hold Times

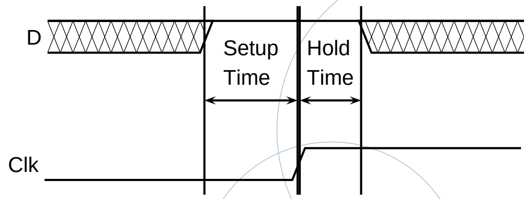
Flip-flops and latches have two types of timing requirements:

Setup time: D input must be stable some time before the clock arrives

Hold time: D input must remain stable some time after the clock has arrived

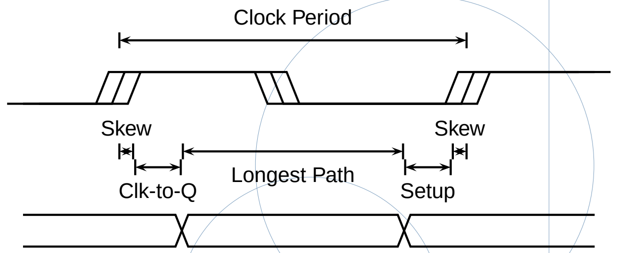
Setup & Hold Times

For an (edge-sensitive) flip-flop



Synchronous System Timing

Budgeting time in a typical synchronous design



Digital Systems

Typical System Architecture

Most large digital systems consist of

Datapath

Arithmetic units (adders, multipliers)

Data-steering (multiplexers)

Memory

Places to store data across clock cycles

Memories, register files, etc.

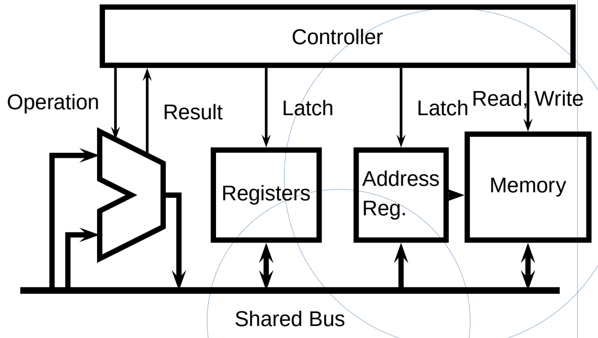
Control

Interacting finite state machines

Direct how the data moves through the datapath

Typical System Architecture

Primitive datapath plus controller



Implementing Digital Logic

Discrete logic chips
- NAND gates four to a chip and wire them up (e.g., TTL)

Programmable Logic Arrays (PLAs)
- Program a chip containing ANDs feeding big OR gates

Field-Programmable Gate Arrays (FPGAs)
- Program lookup tables and wiring routes

Application-Specific Integrated Circuit (ASICs)
- Feed a logic netlist to a synthesis system
- Generate masks and hire someone to build the chip

Full-custom Design
- Draw every single wire and transistor yourself
- Hire someone to fabricate the chip or be Intel

Implementing Digital Logic

Discrete logic is dead
- Requires too many chips

PLAs
- Nice predicable timing, but small and limited

FPGAs
- High levels of integration, very convenient
- Higher power and per-unit cost than ASICs

ASICs
- Very high levels of integration, costly to design
- Low power, low per-unit cost, but huge initial cost

Full-custom
- Only cost-effective for very high-volume parts
- E.g., Intel microprocessors

Digital Logic in Embedded Systems

Low-volume products (1000s or less) typically use FPGAs

High-volume products usually use ASICs

Non-custom logic usually implemented using application-specific standard parts

- Processor chipsets
- Graphics controllers
- PCI bus controllers
- USB controllers
- Ethernet interfaces