CS 280A1/250: FOUNDATION OF COMPUTER SYSTEMS
[MEMORY]

Slicing and dicing time
The smaller you cut
The faster you are
While doing more
Cut too small
And you run into the Physics wall
With legions of light and heat
Conspiring to lead a mutiny
Use the Goldilocks rule
And it's all cool

SHRIDEEP PALICKARA
Computer Science
Colorado State University

Frequently asked questions from the previous class survey

- Idempotent: \( x \text{ And } x = x \quad x \text{ Or } x = x \)
- Xor: Can there be more than 2 inputs?
- Why can't we just use Or, And, and Not? Why must we use Nand?
- What is De Morgan's Law’s relationship to the fundamental theorem?
- Why is Nand preferred over Nor when they are both “universal”?  
  - Delay and size
Topics covered in this lecture

- CPU clocks
- Combinational logic
- Sequential logic
- Data flip flop

**MEMORY**

If only there could be an invention that bottled up a memory, like scent. And it never faded, and it never got stale. And then, when one wanted it, the bottle could be uncorked, and it would be like living the moment all over again.

Daphne du Maurier, Rebecca
Memory Management: Topics we will cover over the next few lectures

- Data flip flop
- The speed differential across the memory hierarchy
- Why caches are needed
- Registers and L1, L2, and L3 caches
- Main memory
- Memory addressing

Computer programs use variables, arrays, and objects

- These are abstractions that persist data over time
- Hardware platforms support this ability by offering memory devices that know how to maintain state
- Because evolution gave humans a phenomenal electro-chemical memory system
  - We tend to take for granted the ability to remember things over time
  - However, this ability is hard to implement in classical logic, which is aware of neither time nor state
To get started

- First find a way to
  - Model the progression of time and
  - Endow logic gates with the ability to maintain state & respond to time changes

- Done by introducing a clock and an elementary, time-dependent logic gate that can flip and flop between two stable states:
  - Representing 0 and representing 1

- This gate, called **data flip-flop** (DFF), is the fundamental building block

Data flip flop (DFF)

- Despite its **central role** the DFF is a low-profile, inconspicuous gate

- DFFs are **used implicitly**, as low-level chip-parts embedded deep within other memory devices
  - Unlike registers, RAM devices, and counters, which play prominent roles in computer architectures
TIME

The best thing about the future is that it comes one day at a time.
—Abraham Lincoln

So far …

- We have assumed that chips *respond* to their inputs *instantaneously*: you input 7, 2, and “add” into the ALU, and … poof! the ALU output becomes 9
- In reality, outputs are always *delayed*
Outputs are always delayed, due to at least two reasons

- The inputs don’t appear out of thin air
  - Rather, the signals that represent them **travel** from the outputs of other chips, and this **travel takes time**

- The **computations** that chips perform also take time
  - The more chip-parts the chip has—the more elaborate its logic
    - The more time it will take for the chip’s outputs to **emerge fully formed** from the chip’s circuitry

Time is typically viewed as a metaphorical arrow that progresses relentlessly forward

- This progression is taken to be **continuous**
  - Between every two time-points there is another time-point, and changes in the world can be **infinitesimally small**

- This notion of time, which is popular among philosophers and physicists, is too deep and mysterious for computer scientists
Instead of viewing time as a continuous progression

- We prefer to break it into fixed-length intervals, called cycles
- This representation is discrete
  - Resulting in cycle 1, cycle 2, cycle 3, and so on
- Unlike the continuous arrow of time, which has an infinite granularity, the cycles are atomic and indivisible
- Changes in the world occur only during cycle transitions; within cycles, the world stands still

Of course, the world never stands still

- However, by treating time discretely, we make a conscious decision to ignore continuous change
- We are content to know the state of the world in cycle $n$, and then in cycle $(n+1)$ but during each cycle the state is assumed to be ... 
  - Well, we don’t care
This discrete view of time serves two important design objectives

- First, it can be used for neutralizing the randomness associated with communications and computation time delays.
- Second, it can be used for synchronizing the operations of different chips across the system.

Discrete time representation

![Diagram showing discrete time representation with examples of inputs and outputs, and a clock cycle with time delays.](image-url)
Typically, the cycles are realized by an oscillator that alternates continuously between two phases labeled 0−1, low-high, or ticktock.

The elapsed time between the beginning of a tick and the end of the subsequent tock is called a cycle. Each cycle is taken to model one discrete time unit.

The current clock phase (tick or tock) is represented by a binary signal. Using the hardware’s circuitry, the same master clock signal is simultaneously broadcast to every memory chip in the system. In every such chip, the clock input is funneled to the lower-level DFF gates. Where it serves to ensure that the chip will commit to a new state, and output it, only at the end of the clock cycle.
Discrete time representation

Example: Considering the Not gate [1/2]

- When we feed the gate with 1, it takes a while before the gate's output stabilizes on 0.
- However, since the cycle duration is — by design — longer than the time delay.
  - When we reach the cycle's end, the gate output has already stabilized on 0.
Example: Considering the Not gate

- Since we probe the state of the world only at cycle ends, we don’t get to see the interim time delays.
  - Rather, it appears as if we fed the gate with 0, and poof! the gate responded with 1.
- If we make the same observations at the end of each cycle?
  - We can generalize that when a Not gate is fed with some binary input x, it instantaneously outputs Not (x).

Goldilocks & The Clock Cycle
For this scheme of discrete time representations to work?

- The cycle’s length must be longer than the maximal time delays that can occur in the system.
- Indeed, cycle length is one of the most important design parameters of any hardware platform!

The hardware engineer chooses the cycle length that meets two design objectives.

- On the one hand, the cycle should be sufficiently long to contain, and neutralize, any possible time delay.
- On the other hand, the shorter the cycle, the faster the computer.
  - If things happen only during cycle transitions, then obviously things happen faster when the cycles are shorter.
  - To sum up: the cycle length is chosen to be slightly longer than the maximal time delay in any chip in the system.
A cutting-edge view of the clock cycle

- Following the tremendous progress in switching technologies, we are now able to create cycles as tiny as a billionth of a second
  - Achieving remarkable computer speed

- The new 12th Gen Intel® Core™ i9
  - 5.5 GHz (announced earlier this year)
Turbo Boost

- Several Intel chips support this
- Enhance clock speed **dynamically** based on *thermal headroom* in the processor to the maximum safe level
  - For example, number of cores are accounted for
    - Intel® Turbo Boost Technology

Overclocking

- AMD has always supported this feature
- Possible on Intel CPUs with a “K” in the name (signifying unlocked)
- Signifies an unlocked **“multiplier”** when used in tandem with a motherboard chipset that supports overclocking
  - Make sure you have proper cooling!
    - Probably use **water cooled** systems
    - Overclocking typically not done on laptops
Overclocking: Disadvantages

- **CPU heat**
  - You are using the CPU to do something you aren't supposed to do
    - Outside typical operating conditions
  - Introduces wear and tear
    - Shelf-life reduces

- While complete failure does not typically occur
  - Leads to a lot of instability

---

The Data Flip flop
Memory

- Memory chips are designed to “remember”, or store, information over time.
- The low-level devices that facilitate this storage abstraction are named flip-flop gates, of which there are several variants.

Data flip flop

- DFF’s interface includes a single-bit data input and a single-bit data output.
- The DFF has a clock input that feeds from the master clock’s signal.
- Taken together, the data input and the clock input enable the DFF to implement the simple time-based behavior \( out(t) = in(t-1) \)
  - Where \( in \) and \( out \) are the gate’s input and output values.
  - \( t \) is the current time unit.
Like Nand gates, DFF gates lie deep in the hardware hierarchy

- All the memory chips in the computer—registers, RAM units, and counters—are based, at bottom, on DFF gates
- All these DFFs are connected to the same master clock, forming a huge distributed “chorus line”

At the end of each clock cycle, the outputs of all the DFFs in the computer commit to their inputs from the previous cycle

At all other times, the DFFs are latched, meaning that changes in their inputs have no immediate effect on their outputs
DFF conduction operations

- This conduction operation affects any one of the system’s numerous DFF gates many times per second
  - Depending on the computer’s clock frequency

- Hardware implementations realize the time dependency using a dedicated clock bus that feeds the master clock signal simultaneously to all the DFF gates in the system.
Combinational

- Chips encompassing the elementary logic gates we have discussed so far were designed to respond only to changes that occur during the current clock cycle.
- Such chips are called time-independent, or combinational.
- The combinational name alludes to the fact that these chips respond only to different combinations of their input values.
- While paying no attention to the progression of time.

Sequential circuits

- Chips that are designed to respond to changes that occurred during previous time units (and possibly during the current time unit as well) are called sequential, or clocked.
- The most fundamental sequential gate is the DFF, and any chip that includes it, either directly or indirectly, is also said to be sequential.
Sequential logic configuration

- The main element in this configuration is a set of one or more chips that include DFF chip-parts, either directly or indirectly.
- These sequential chips may also interact with combinational chips.
- The **feedback loop** enables the sequential chip to respond to inputs and outputs from the previous time unit.
In combinational chips, where time is neither modeled nor recognized

- The introduction of feedback loops is problematic
- The output of the chip would depend on its input, which itself would depend on the output, and thus the output would depend on itself

However ...

- There is no difficulty in feeding outputs back into inputs, as long as the feedback loop goes through a DFF gate
- The DFF introduces an inherent time delay so that the output at time $t$ does not depend on itself but rather on the output at time $t - 1$
Suppose we instruct the ALU to compute $x + y$ [1/2]

- $x$ is the output of a nearby register, and $y$ is the output of a remote RAM register.
- Because of physical constraints like distance, resistance, and interference, the electric signals representing $x$ and $y$ will likely arrive at the ALU at different times.

Suppose we instruct the ALU to compute $x + y$ [2/2]

- However, being a combinational chip, the ALU is insensitive to the concept of time.
  - It continuously and happily adds up whichever data values happen to lodge at its inputs.
- Thus, it will take some time before the ALU’s output stabilizes to the correct $x + y$ result.
Until then, the ALU will generate garbage

- How can we overcome this difficulty?
- Well, if we use a discrete representation of time, we simply don’t care
- All we have to do is set the clock cycle duration effectively
  - Slightly longer than the time it takes a bit to travel the longest distance from one chip to another, plus the time it takes to complete the most time-consuming within-chip calculation
  - This way, we are guaranteed that by the end of the clock cycle, the ALU’s output will be valid

DFF gate

[Diagram of DFF gate with input and output signals and timing diagram showing load, in, and out values over time]
DFF gate

- A DFF gate has a single-bit data input, a single-bit data output, a clock input, and a simple time-dependent behavior:
  - \( \text{out}(t) = \text{in}(t-1) \)

- Usage:
  - If we put a one-bit value in the DFF's input, the DFF's state will be set to this value, and the DFF's output will emit it in the next time unit.

DFF gate

- A DFF gate is designed to be able to “flip-flop” between two stable states, representing 0 and representing 1.

- This functionality can be implemented in several different ways, including ones that use **Nand** gates only.

- The Nand-based DFF implementations are elegant, yet intricate.
A data flip-flop: Under the hood

The contents of this slide-set are based on the following references