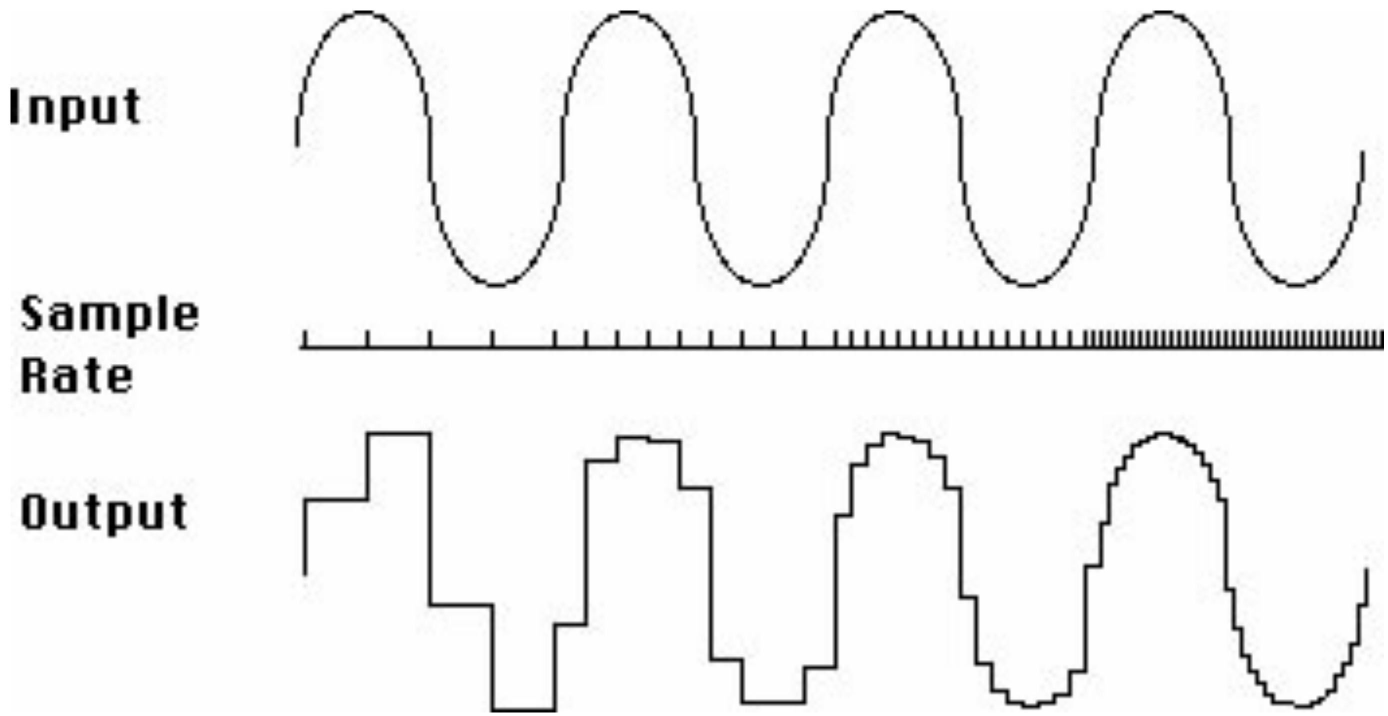
Sampling Inputs, Latency, and Input Conditioning

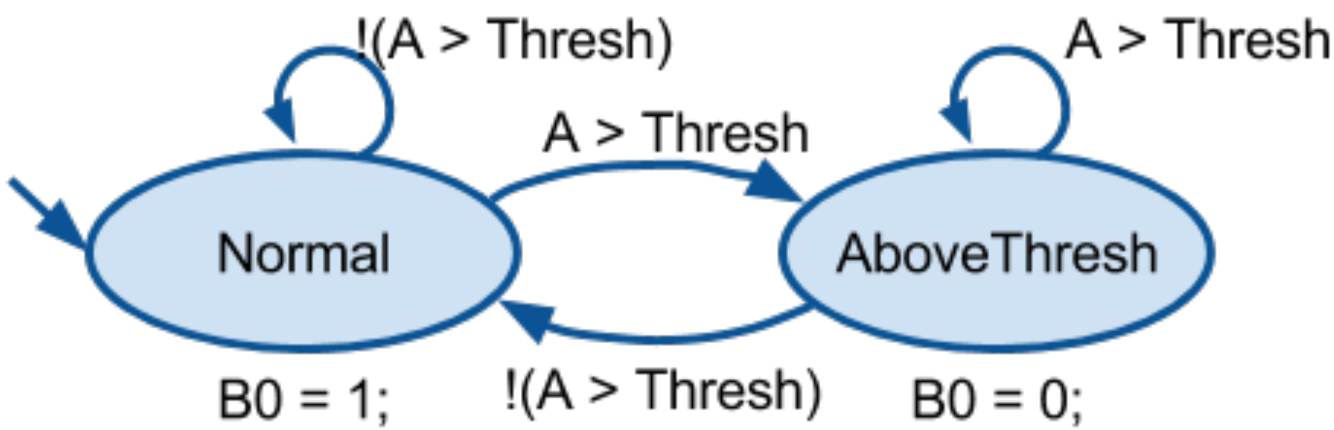
Sampling

- Reading a sensor at a specified period
  - The period is called the *sampling rate*



Speaker Example

SpeakerProtect
Period: __?__ms
const unsigned char Thresh = 85;



- The speaker will be damaged if the input level exceeds 85 for a long period of time
- What is a reasonable sampling rate for the audio level?

Issues / Constraints

- Sampling too slow
  - Damage may occur when the audio input level spikes between samples
- Sampling too fast
  - The microcontroller cannot complete its Tick() function before the next sample

Sampling Rate Criteria

- Sample as fast as possible, to maximize accuracy
- Sample as slow as possible, to conserve processor time
- Sample fast enough to provide adequate response time
- Sample slow enough that noise doesn't dominate the input signal
- Sample at a rate that is a multiple of the control algorithm frequency to minimize jitter

A Practical Approach

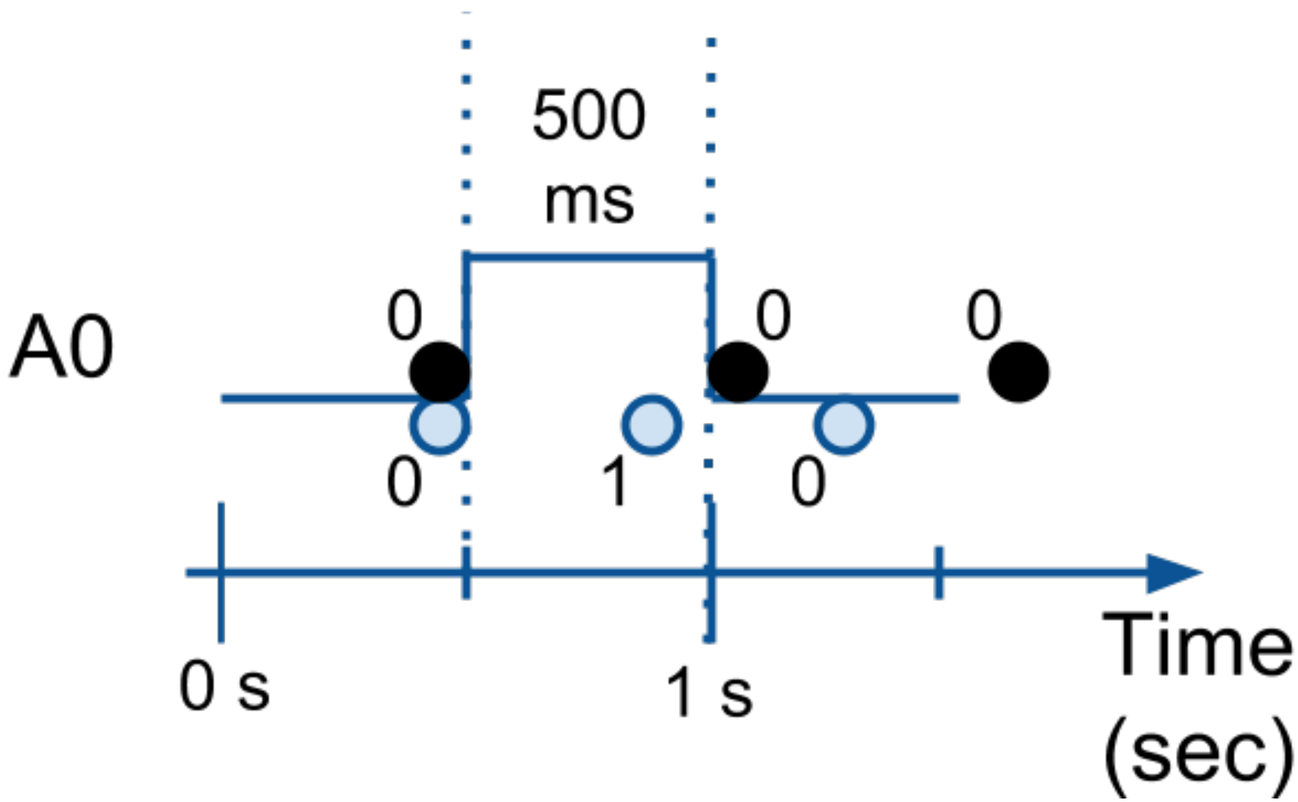
1. Measure sensor characteristics.
2. If there is noise in the input, select the algorithm that will be used to filter the data.
3. Compute the lower and upper bound for sampling rates based on function alone.
4. Identify the trade-offs between using the lower and upper bound rates.
5. Prioritize the trade-offs to determine a suitable sampling rate that is between the computed lower and upper bounds.

Challenges/Constraints

- Don’t miss events that occur
- Recognize distinct events

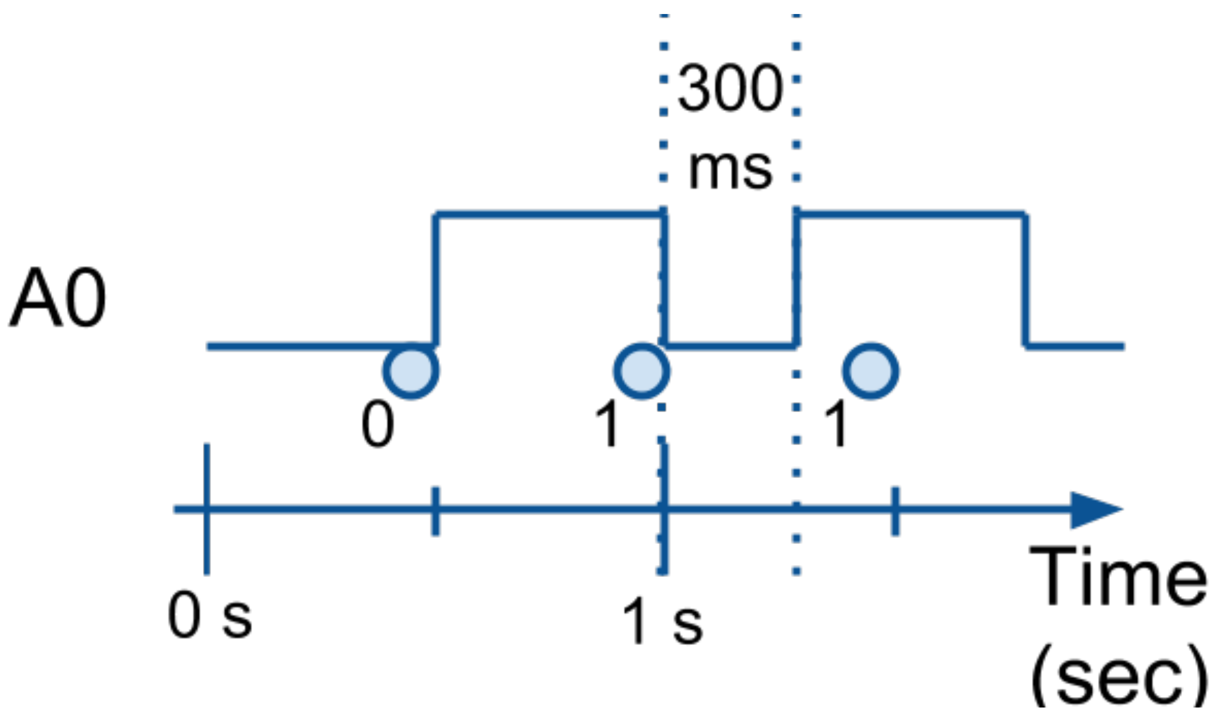
(Don't Miss) Input Events

- A sampling interval < 500ms <u>cannot</u> miss a 500ms pulse



Recognizing Distinct Events

- Two consecutive samples of 1
  - One long event?
  - Two (more?) short events?



The Reality of Physical Phenomena

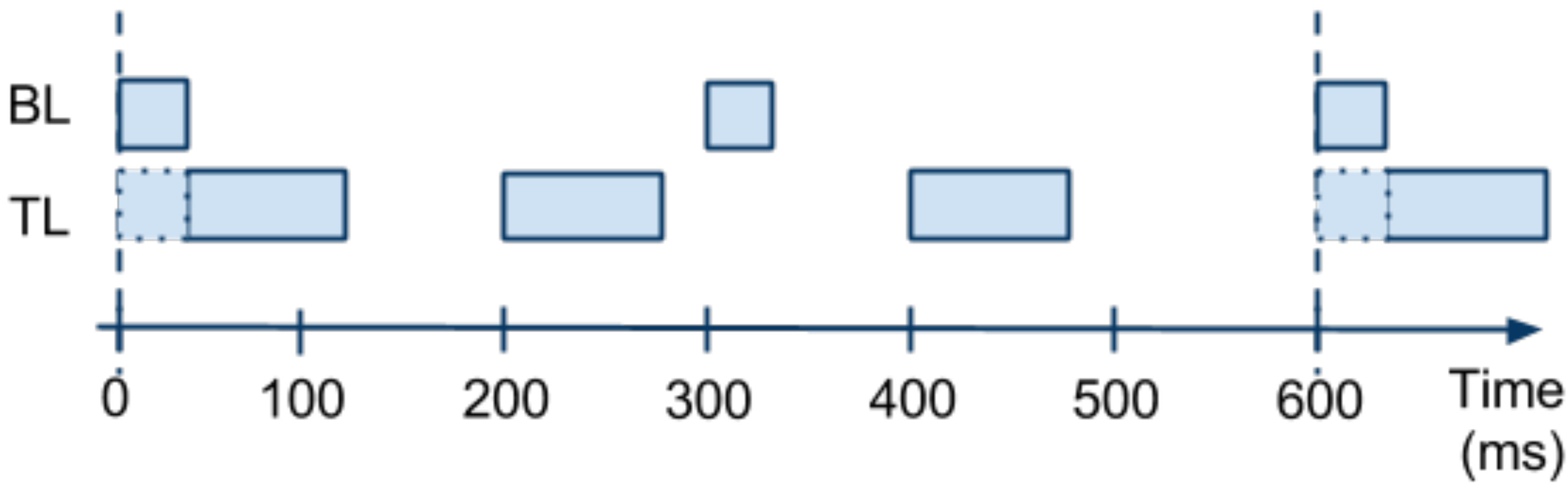
- Rarely periodic
  - Will your heartbeat be the same for your whole life?
- The length of a detectable event may vary
  - Does every button press last 500 ms?
    - Infant vs. adolescent vs. adult vs. geriatric
    - People with physical disabilities, etc.
- Some phenomena require a sampling rate that exceeds today's computational capabilities
  - Major interdisciplinary engineering challenges
  - E.g., Large Hadron Collider

CS/EE 120B Simplification

- Minimum/Maximum sampling intervals will be given as part of project descriptions, e.g.:
  - A button press lasts 500 ms
  - The minimum time between presses is 1000ms
- Given this information, your job is to pick a synchSM period that works correctly
  - There may be multiple solutions
    - 500 ms, 250 ms, 100 ms, 50 ms, 25 ms, 10 ms, 5 ms, 1 ms, …
  - What to do?

Utilization

- The fraction of time that the microcontroller is executing tasks



Our Strategy: Minimize Utilization

- Minimize Utilization
  - Choose the largest sampling period that satisfies system requirements!
    - $\boxed{\text{500 ms}}$, 250 ms, 100 ms, 50 ms, 25 ms, 10 ms, 5 ms, 1 ms, …

- Why?
  - Minimize energy consumption
    - Sleep() between ticks
  - Free up microcontroller time to execute more tasks
    - For concurrent synchSMs
    - CS 122A covers periodic task scheduling

Latency

- Time between an input event and the output event that it triggers

- Examples:
  - Doorbell
  - Light switch
  - Vehicles
  - Video game (lag)

Latency vs. Utilization Tradeoff

Reduce latency => Shorter Period => Higher Utilization

Minimize Utilization => Longer Period => Higher Latency

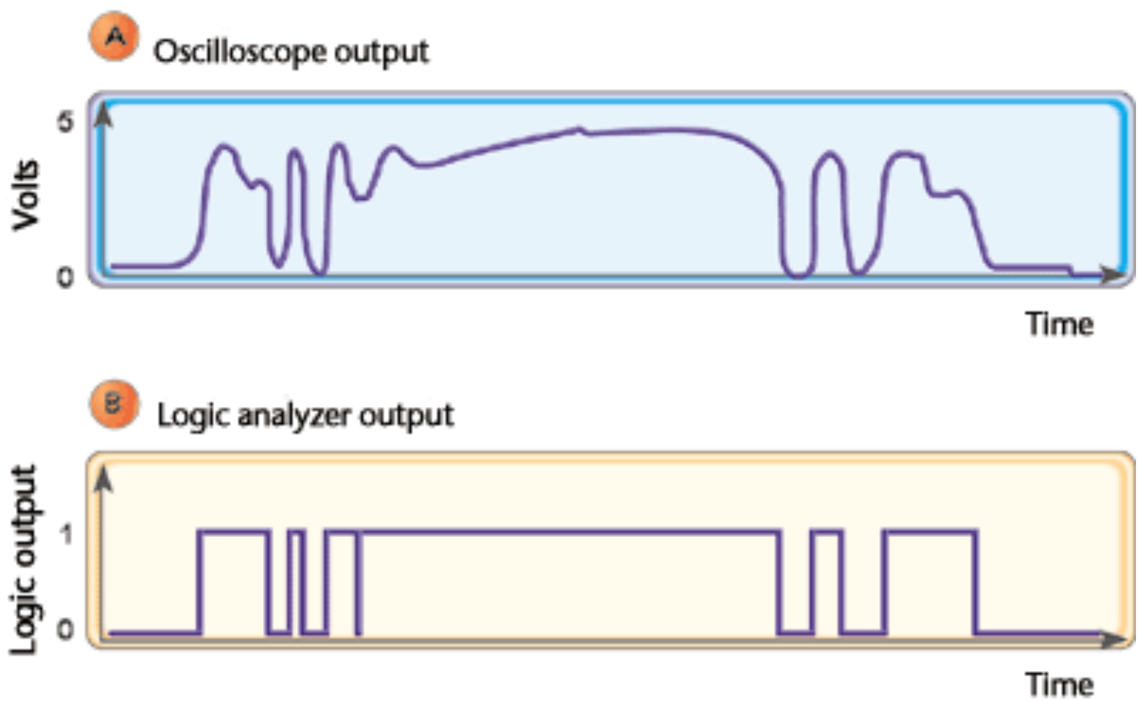
- <u>Our Approach</u>
  - Latency is given as a constraint
  - Minimize Utilization without violating the latency (or any other) constraint(s)

Constraint Table (Doorbell Example)

Timing specification	Constraint
Minimum press length 400 ms	Period should be < 400 ms
Minimum separation time between a button release and a button press: 500 ms	Period should be < 500 ms
Maximum latency between press and bell: 100ms	synchSM period should be < 100 ms; state sequence should ensure latency <= 100 ms
Bell rings for 1 sec	Period should evenly divide 1000 ms
Minimize processor utilization	Period should be as large as possible

Input Conditioning

- Sensors are not perfect
- Buttons and switches bounce when pressed



Debouncing

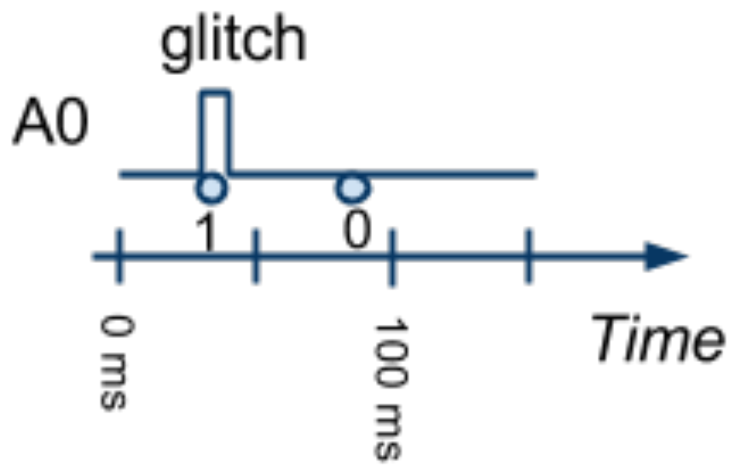
- Ignore the bouncing so that you register one button press
  - Modern buttons bounce for 10-20 ms
  - A period of >= 50 ms does the job

One More Constraint for the Doorbell

Timing specification	**Constraint**
Minimum press length 400 ms	Period should be < 400 ms
Minimum separation time between a button release and a button press: 500 ms	Period should be < 500 ms
Maximum latency between press and bell: 100ms	synchSM period should be < 100 ms; state sequence should ensure latency <= 100 ms
Bell rings for 1 sec	Period should evenly divide 1000 ms
Debounce by sampling no faster than every 50 ms	Period should be >= 50 ms
Minimize processor utilization	Period should be as large as possible

Glitches

- Short, temporary signal spikes



- Given a sensor reading of ‘1’, how can you tell if it is accurate or a glitch?

Filtering

- Strategies to ignore spurious input events (e.g., glitches)
  - Detect a ‘1’ for two (or more) consecutive samples before confirming it to be legitimate
  - Choose a sampling interval longer than the minimum glitch duration

Filtering Drawbacks and Limitations

- Increases latency
  - Multiple samples are needed before we can determine that an input event has occurred and an action is taken
- Cannot guarantee perfection
  - Legitimate input vs. sampling two glitches?
  - (Probability…)
- (Analog) hardware-based solutions work well in practice, but are beyond the scope of CS/EE 120B