

Delta Timing Review

CSE 1302

Today's Outline

- Delta timing
- Pushbutton debouncing
- Using SerialComm

Review of Delta Timing

```
while (true)
```



```
  now = millis()
```

```
  if (now >= loopEndTime) then
```

```
    loopEndTime += deltaTime
```

```
    do some work
```

```
    output results
```

```
  end if
```

```
end while
```



What if work takes a while?

```
while (true)
```

```
  now = millis()
```

```
  if (now >= loopEndTime) then
```

```
    loopEndTime += deltaTime
```

```
     do some work
```

```
  end if
```

```
end while
```

Especially if work sometimes takes longer than
deltaTime!

What if work takes a while?

```
while (true)
```

```
  now = millis()
```

```
  if (now >= loopEndTime) then
```

```
    loopEndTime += deltaTime
```

```
    do some work
```

```
  end if
```

```
end while
```



Alternative approach

while (true)

 now = millis()

 if (now >= loopEndTime) then

 ➡ loopEndTime = loopEndTime + deltaTime

 OR

 ➡ loopEndTime = now + deltaTime

 do some work

 end if

end while

Alternative approach

```
while (true)
```

```
    now = millis()
```

```
    if (now >= loopEndTime) then
```

```
        loopEndTime = now + deltaTime
```

```
        do some work
```

```
    end if
```

```
end while
```



Which Update Should I Use?

loopEndTime += deltaTime

Keeps the average rate exact

If work runs long, the next iterations run back-to-back to catch up

If work always takes longer than deltaTime, it never catches up: it falls further behind and the if is true on every pass

Good when the rate matters: sampling, clocks, counting time

loopEndTime = now + deltaTime

Always at least deltaTime between iterations

If work runs long, the whole schedule slides later (drift)

If work always takes longer than deltaTime, it just runs back-to-back

Good when the gap matters more than the long-run rate

The Same Thing in Arduino C

```
const unsigned long beatPeriod = 1000;    // ms
```

Keeps the average rate

```
unsigned long lastBeatTime = 0;

void loop() {
  unsigned long now = millis();
  if (now - lastBeatTime >= beatPeriod) {
    lastBeatTime += beatPeriod;
    Serial.print("beat");
  }
}
```

Keeps a minimum gap

```
unsigned long nextBeatTime = 0;

void loop() {
  unsigned long now = millis();
  if (now >= nextBeatTime) {
    nextBeatTime = now + beatPeriod;
    Serial.print("beat");
  }
}
```

Both versions are from Studio 3

loop() is the while (true): Arduino calls it over and over

Globals like lastBeatTime keep their values between passes of loop()

Think Like a Finite-State Machine

```
while (true)
  ➡ if (millis() >= loopEndTime) then
      loopEndTime += deltaTime
      do some work
    end if
  end while
```

Do some (but not all) of the work

Remember “state” information (in one or more variables)

Inside delta time conditional if, add switch statement

Use the Right Type for Time

- `millis()` returns an unsigned long (4 bytes on the Uno)
 - 0 to 4,294,967,295 ms (about 49.7 days), then it wraps back to 0
- An int on the Uno is only 2 bytes
 - Largest value is 32,767, so an int time variable breaks after about 33 seconds
- Rollover-safe version (Studio 3's first version): compare elapsed time

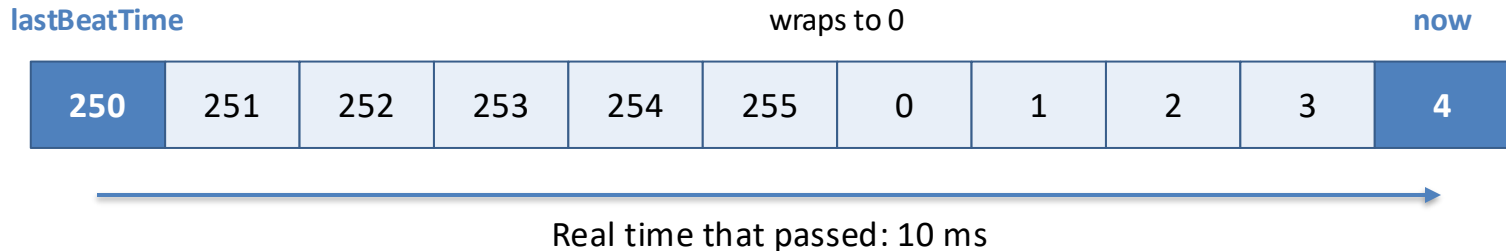
```
unsigned long lastBeatTime = 0;    // not int!

if (now - lastBeatTime >= beatPeriod) {
    lastBeatTime += beatPeriod;
    do some work
}
```

 - Why is this safe when `millis()` wraps back to 0? Next slide

Why Subtraction Is Rollover Safe

Pretend millis() were 8 bits (0 to 255, then it wraps to 0), with beatPeriod = 10



Subtraction: $\text{now} - \text{lastBeatTime}$

4 - 250 would be -246, but unsigned math can't go negative: it wraps around by 256

$-246 + 256 = 10$, exactly the time that passed, so the beat happens on time

Comparing times: $\text{now} \geq \text{nextBeatTime}$

Beat at 250, so $\text{nextBeatTime} = 250 + 10 = 260$, which wraps to 4

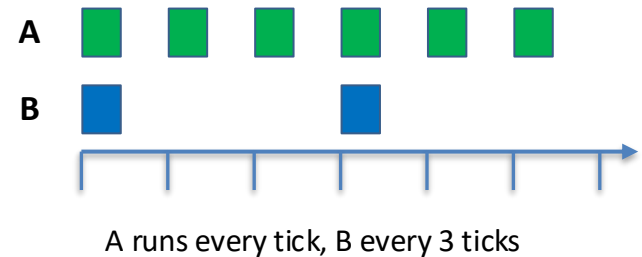
At $\text{now} = 251$, $251 \geq 4$ is already true, so it fires 9 ms early (and keeps firing until now wraps too)

The subtraction wraps the same way the clock does, so $\text{now} - \text{lastBeatTime}$ is always the real elapsed time

Real millis() is 32 bits, so the wrap happens after about 49.7 days, but the idea is the same

Several Things at Once

```
while (true)
  now = millis()
  if (now >= taskAEndTime) then
    taskAEndTime += taskADelta
    do task A
  end if
  if (now >= taskBEndTime) then
    taskBEndTime += taskBDelta
    do task B
  end if
end while
```

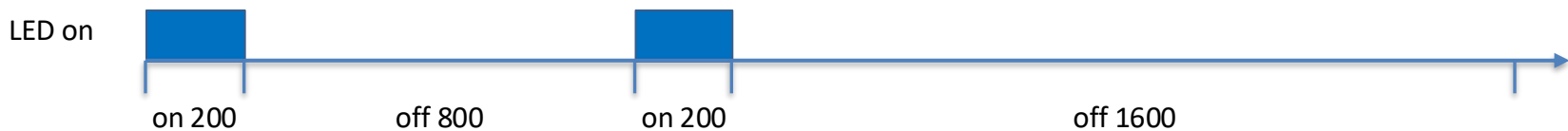


Each task has its own end time (and its own delta)

Neither task waits for the other, so they run independently

Looping Without a Loop

```
pattern = {200, 800, 200, 1600} // ms for each step
index = 0                        // global: remembers where we are
while (true)
    now = millis()
    if (now >= stepEndTime) then
        toggle LED
        stepEndTime += pattern[index]
        index = (index + 1) % 4
    end if
end while
```



No for loop: the global index replaces the loop counter, and loop() does the repeating

Each step can have its own time, like the Cricket (Assignment 3): on for BLINK_DURATION, then off for period - BLINK_DURATION

Common Delta-Timing Mistakes

Work outside the if

Code outside the if runs on every pass of loop(), which can be thousands of times between ticks

Calling millis() more than once

Each call can return a different value. Call it once per pass and save it in now

Starting the end time at 0

The first iteration runs right away. If that's not what you want, set it from millis() in setup()

Storing times in an int

An int overflows after about 33 seconds. Use unsigned long for anything that holds a millis() value

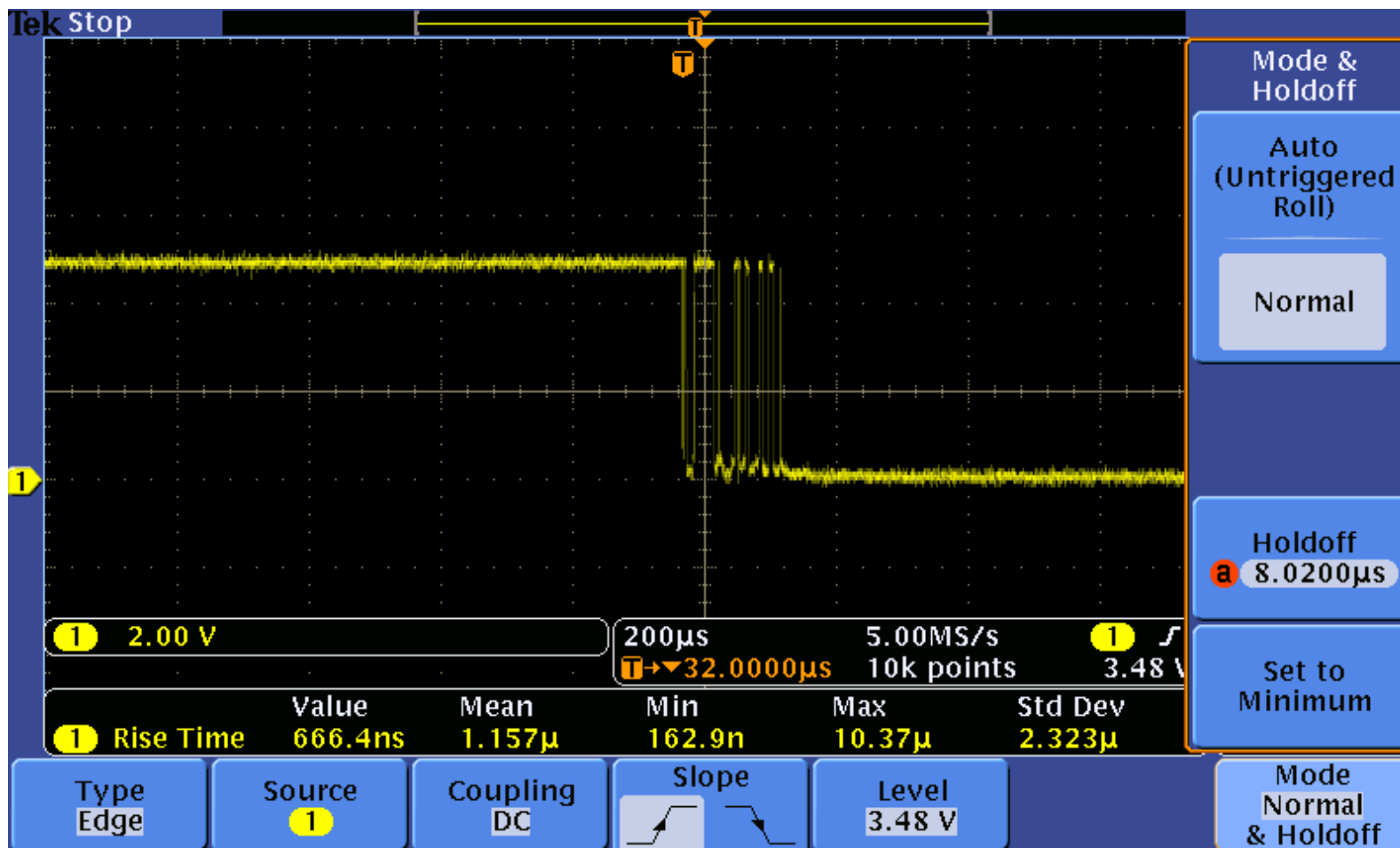
Back to Pushbutton Debouncing

- Switch between input pin and GND
 - Input goes LOW when switch is pressed
 - Input goes HIGH when switch is not pressed

```
void setup() {  
    pinMode(pin, INPUT_PULLUP);  
}
```

```
void loop() {  
    inputVal = digitalRead(pin);  
}
```

Let's See What Happens



Switch “Debouncing”

Read switch state

Wait enough time for switch to quit bouncing 

Read switch state again

if two switch states agree

 Done

else

 Start over

Delta-time Debouncing

```
while (true)
  now = millis()
  if (now >= dloopEndTime) then
    dloopEndTime = dloopEndTime + debounceTime
    switch (whichRead)
      case 0: read1 = digitalRead(pin)
              break
      case 1: read2 = digitalRead(pin)
              if (read1 == read2)
                inputVal = read1
              endif
              break
    whichRead = !whichRead
  end if
  if (now >= otherLoopEndTime ) then ...
```

whichRead is the state variable: a two-state FSM

Quiz Time!

True or False?

Delta Timing and `delay()` accomplish the same thing.