TFB2093 Internet-of-Things

Practical 05: IoT Programming – Understanding Timing (Arduino Uno in Tinkercad)

Objectives

- Demonstrate timing using `delay()`, `millis()`, and `micros()`.
- Implement a **non-blocking** LED blink using `millis()`.
- Record (print) sensor data at timed intervals and compare results.
- Measure function execution time using `micros()`.

---

0) Lab Setup (Tinkercad)

Components (all available in Tinkercad Circuits): - Arduino Uno R3 (or use the built-in **LED_BUILTIN** on pin 13) - 1 × LED (any color) - 1 × 220 Ω resistor - Optional sensor for reading (use **Potentiometer** → connect to **A0**)

Wiring (if using an external LED): - LED **anode** → **digital pin 8** through a 220 Ω resistor. - LED **cathode** → **GND**. - (Alternative: skip wiring and just use `LED_BUILTIN` on **pin 13**.)

Serial Monitor: - In Tinkercad, start the simulation and open the **Serial Monitor** (9600 bps) to see outputs.

---

1) Compare `delay()`, `millis()`, and `micros()`

You will run **three short sketches** and observe behavior in the Serial Monitor.

1A) Elapsed time using `delay()` (blocking)

> Prints a message every 1000 ms using `delay()`. Notice how nothing else can run during the delay.

```
// 1A_delay_demo.ino (Tinkercad-only)
// Prints timestamp once per second using delay() — blocks loop.

void setup() {
  Serial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT); // you can watch the built-in LED too
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(100);
  digitalWrite(LED_BUILTIN, LOW);

  unsigned long t = millis();
  Serial.print("[delay demo] t(ms)=");
  Serial.println(t);

  delay(1000); // BLOCKING: nothing else happens here
}
```

Observe: Timestamps increase by ~1000 ms steps. Try adding extra code (e.g., reading A0) — it will also be delayed.

1B) Elapsed time using `millis()` (non-blocking interval)

> Prints a message every 1000 ms **without** using `delay()`.

```
// 1B_millis_demo.ino (Tinkercad-only)
// Prints once per second using millis() — non-blocking.

const unsigned long PRINT_INTERVAL = 1000; // ms
unsigned long lastPrint = 0;               // ms

void setup() {
  Serial.begin(9600);
}

void loop() {
  unsigned long now = millis();

  // non-blocking interval check
  if (now - lastPrint >= PRINT_INTERVAL) {
    lastPrint = now;
    Serial.print("[millis demo] t(ms)=");
    Serial.println(now);
  }

  // Other work can run freely here (no blocking):
  int val = analogRead(A0); // if a pot is connected, twist it to see live
values
  (void)val; // suppress unused warning
}
```

Observe: Serial time is ~1000 ms apart **while** analog reads still run continuously.

1C) Elapsed time using `micros()` (microsecond resolution)

> Prints a message about every 50,000 µs (0.05 s). Resolution ~4 µs on Uno (approx.).

```
// 1C_micros_demo.ino (Tinkercad-only)
// Prints roughly every 50,000 microseconds using micros().

const unsigned long PRINT_US = 50000; // 50 ms
unsigned long lastUs = 0;

void setup() {
  Serial.begin(9600);
}

void loop() {
  unsigned long nowUs = micros();
  if ((unsigned long)(nowUs - lastUs) >= PRINT_US) {
    lastUs = nowUs;
    Serial.print("[micros demo] t(µs)=");
    Serial.println(nowUs);
  }

  // do other quick work if needed
}
```

Observe: Faster timestamps, in **microseconds**. Useful for short/fast measurements.

Summary of 1A–1C: - `delay(ms)` is **blocking**. - `millis()` gives **ms** since power-up (good for scheduling tasks without blocking). - `micros()` gives **µs** since power-up (good for timing very fast operations).

---

2) Non-Blocking LED Blink using `millis()`

Blink every 1 second **without** `delay()`. Keep looping to do other tasks concurrently.

```
// 2_nonblocking_blink.ino (Tinkercad-only)

const int LED_PIN = 8;                 // or use LED_BUILTIN
const unsigned long BLINK_INTERVAL = 1000; // ms

unsigned long lastToggle = 0;
bool ledState = false;

void setup() {
  Serial.begin(9600);
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  unsigned long now = millis();
  // 1) Non-blocking LED toggle
  if (now - lastToggle >= BLINK_INTERVAL) {
    lastToggle = now;
    ledState = !ledState;
    digitalWrite(LED_PIN, ledState ? HIGH : LOW);
    Serial.print("[blink] toggled at t(ms)=");
    Serial.println(now);
  }

  // 2) Other work continues smoothly (e.g., sensor read)
  int sensor = analogRead(A0); // Potentiometer → A0
  // Print occasionally (every 200 ms) without blocking
  static unsigned long lastSense = 0;
  if (now - lastSense >= 200) {
    lastSense = now;
    Serial.print("A0=");
    Serial.println(sensor);
  }
}
```

Expected outcome: LED blinks steadily; Serial shows both blink events and sensor prints **without pausing** the loop.

---

3) Blink LED with Increasing Speed using `millis()`

> Start at 1000 ms interval; **every 5 seconds** reduce interval by 200 ms down to a minimum (e.g., 200 ms), then reset.

```
// 3_speedup_blink.ino (Tinkercad-only)

const int LED_PIN = 8;                      // or LED_BUILTIN
unsigned long blinkInterval = 1000;         // starts at 1000 ms
const unsigned long MIN_INTERVAL = 200;     // ms
const unsigned long STEP_EVERY = 5000;      // reduce interval every 5 s
const unsigned long STEP_SIZE = 200;        // reduce by 200 ms

unsigned long lastToggle = 0;    // for LED blink
unsigned long lastStep   = 0;    // for speed change
bool ledState = false;

void setup() {
  Serial.begin(9600);
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  unsigned long now = millis();
  // Blink logic
  if (now - lastToggle >= blinkInterval) {
    lastToggle = now;
    ledState = !ledState;
    digitalWrite(LED_PIN, ledState ? HIGH : LOW);
  }

  // Interval reduction every 5 seconds
  if (now - lastStep >= STEP_EVERY) {
    lastStep = now;
    if (blinkInterval > MIN_INTERVAL + STEP_SIZE) {
      blinkInterval -= STEP_SIZE;
    } else {
      blinkInterval = 1000; // reset to start speed
    }
    Serial.print("[speed] new interval(ms)=");
    Serial.println(blinkInterval);
  }
}
```

Tip: Watch the LED blink faster in steps; Serial prints the new interval.

---

4) Use `micros()` to Measure Function Execution Time

> Measure how long a computation takes. Example: sum numbers or take multiple analog readings.

```
// 4_execution_time_micros.ino (Tinkercad-only)

unsigned long measureWork() {
  // Example workload: sum 0..999 and 10 analogRead() calls
  volatile unsigned long sum = 0; // volatile prevents optimization removal
  for (int i = 0; i < 1000; i++) {
    sum += i;
  }
  for (int k = 0; k < 10; k++) {
    sum += analogRead(A0);
  }
  return sum; // just to use it
}

void setup() {
  Serial.begin(9600);
}

void loop() {
  unsigned long t0 = micros();
  unsigned long out = measureWork();
  unsigned long t1 = micros();
  unsigned long dt = (unsigned long)(t1 - t0);

  Serial.print("[micros timing] work result=");
  Serial.print(out);
  Serial.print(", took(μs)=");
  Serial.println(dt);

  delay(500); // just to slow down printing; timing measured uses micros()
}
```

Expected: You'll see microsecond-scale durations. Change loop sizes to see how time grows.

---

5) Discussion & Short Answers

Q1. What is the advantage of using `millis()` over `delay()`?
A: `millis()` enables **non-blocking** timing so other tasks (sensor reads, comms, logic) can run concurrently. `delay()` freezes the CPU for the delay period.

Q2. When would you prefer `micros()`?
A: When measuring **very short** events (microsecond precision): pulse widths, tight performance checks, or fast sampling intervals.

Q3. What happens when `millis()` overflows after ~50 days?
A: `millis()` (an `unsigned long`) **rolls over** to 0. Correct code uses subtraction with unsigned arithmetic, e.g., `if (millis() - t0 >= interval)` which safely handles rollover.

---

6) Guided Exercises (to submit screenshots of Serial Monitor & circuit)

1. **Run 1A, 1B, 1C** and capture 10 lines of output each. Comment on blocking vs non-blocking behavior.
2. Modify **2_nonblocking_blink** to print A0 only when the LED turns **ON**.
3. In **3_speedup_blink**, change the minimum interval to **100 ms** and the step size to **100 ms**; plot interval vs. time manually from Serial logs.
4. In **4_execution_time_micros**, double the loop bounds (e.g., 2000 and 20). Report the new execution time and compute the ratio vs original.

---

7) Troubleshooting Checklist

- **Nothing prints?** Ensure `Serial.begin(9600);` in `setup()` and Serial Monitor is open at 9600.
- **LED not blinking?** Check pin number matches wiring. Try `LED_BUILTIN` to avoid wiring errors.
- **Odd time jumps?** Remember Serial printing itself takes time; keep print frequency reasonable.
- **Rollover safety:** Always compare time using subtraction: `if (now - last >= interval)`.

8) (Optional) Bonus: Timed Data Logger (no delay)

Sample A0 every 100 ms for 5 seconds and print `t(ms),value` CSV lines you can copy to a spreadsheet.

```
// bonus_logger.ino (Tinkercad-only)

const unsigned long SAMPLE_PERIOD = 100; // ms
const unsigned long DURATION = 5000;     // ms

unsigned long tStart = 0, tLast = 0;
bool started = false;

void setup() {
  Serial.begin(9600);
  Serial.println("t_ms,value");
}

void loop() {
  unsigned long now = millis();
  if (!started) { started = true; tStart = now; tLast = now; }

  if (now - tLast >= SAMPLE_PERIOD) {
    tLast += SAMPLE_PERIOD; // keeps cadence steady
    int v = analogRead(A0);
    Serial.print(now - tStart);
    Serial.print(",");
    Serial.println(v);
  }

  if (now - tStart >= DURATION) {
    while (1) { /* stop */ }
  }
}
```

End of Practical 05

Remember: in embedded systems, **time management** is as important as the logic itself. Master the non-blocking pattern and you'll write responsive, robust IoT code.