COSC 111B **BSCS - 4C**

Laboratory Activity #1: Working with Digital Signals

Member	Grade
Delacerna, Gabrielle	Leader
Bonganay, Lhojen Faith C.	100
Villamor, Lesterjhon	99

Generative AI Sources:

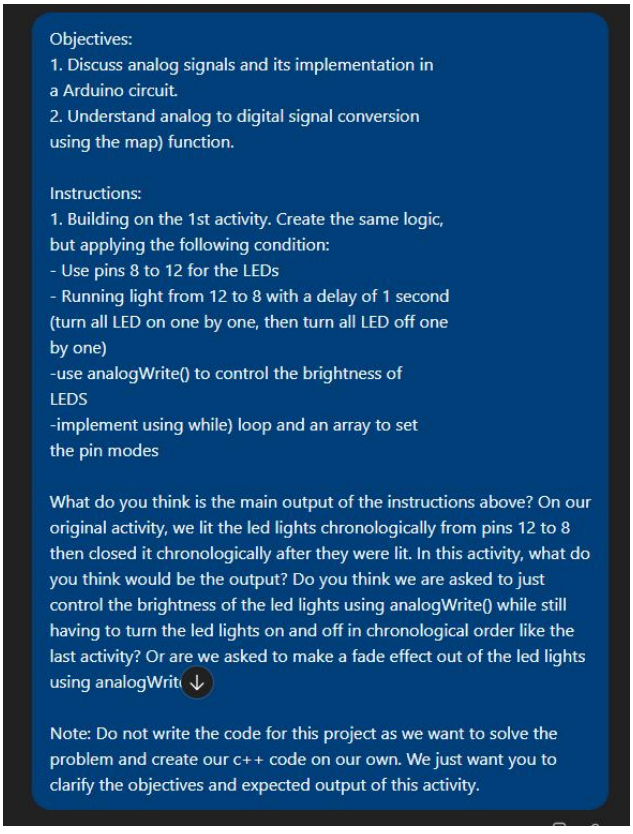
Similar to the last activity, our reliance on AI was light, as we just re-used the code we had for our last activity and tweaked it to fit the recently given lab activity whilst relying on the given lecture/guide provided in our google classroom. Though we still had to ask ChatGPT a few questions as we overthought the output of the activity given that many of our classmates bombarded the group chat (as well as some of them asking us about the output we thought would be correct) with questions that made us rethink our output. Originally, we thought of implementing a nested loop (one for the index looping, and one for the brightness) but figured that this style of output was not the one being asked in the instructions. So for us to further understand and re-do our output, we asked ChatGPT a few questions. Here are the prompts we've subjected to the generative AI we've used:

AI Tool/Applications Used -> ChatGPT

Language Model -> GPT-5

Full Conversation:

https://chatgpt.com/share/68d67f69-4b70-800f-a730-1a6512768f3e



Good question — I see where the confusion is coming from. Let's carefully unpack the instructions and compare them to your first activity.

In your first activity

- LEDs on pins 12 → 8 lit up **one by one** in sequence.
- Then they were **turned off** in the same sequence (chronologically).
- The control was **digital**: `HIGH` (on) and `LOW` (off).

In this new activity

The instructions add **two new requirements** on top of that first one:

1. **Still use pins 8 to 12** → so same LED setup.
2. **Running light from 12 to 8 with a delay of 1 second** → so the on/off sequence is still there, just like before.
   - Turn LEDs on one by one (12, 11, 10, 9, 8).
   - Then turn them off one by one (8, 9, 10, 11, 12).
3. **Use `analogWrite()` to control brightness** → this is the new part.
   - Instead of just *fully on* or *fully off*, the brightness level of each LED should be controlled.
   - This could mean either:
     - Turning them on at a chosen brightness (not necessarily maximum), OR
     - Making them fade in/out (a common analogWrite activity).
4. **Implement with a `while` loop and an array** → this is about *how you code it*, not what the LEDs actually do.

So what's the expected output?

The main behavior is still the sequential ON/OFF pattern (like the first activity), **but now brightness must be involved.**

Most likely:

- Each LED turns on one by one at a certain brightness (using `analogWrite()` instead of `digitalWrite()`), then turns off one by one.
- The fade effect is not explicitly mentioned in the instructions (since they say *"turn all LED on one by one, then turn all LED off one by one"*), so it's probably **not required** unless your instructor wants you to demonstrate the full capability of `analogWrite()`.

✅ **Final clarification:**
You are not replacing the sequential on/off logic with a fade animation. You are simply **redoing the same running light**, but this time with brightness control via `analogWrite()` (instead of just digital on/off).

If they wanted a **fade effect**, the instructions would have explicitly said so (e.g., "fade the LEDs in and out").

Tell me, is the map function really needed in this activity? Or is it something that we just need to learn and understand for ourselves but not really needed in the activity?

That's a very sharp observation 👍

Here's how it breaks down:

What the map() function does

The Arduino `map()` function is just a **math helper**:

```scss
map(value, fromLow, fromHigh, toLow, toHigh)
```

Copy code

It takes a number from one range and scales it to another range.
Example: mapping a potentiometer reading (0–1023) to PWM brightness (0–255).

In your activity

- The instructions don't mention any **input device** (like a potentiometer or sensor).
- You're only told to **run LEDs in sequence** and **use analogWrite() for brightness control.**
- If you're assigning a fixed brightness (say 128 out of 255), you don't need `map()`.

Why the instructions mention it

- The **objective** says: *"Understand analog to digital signal conversion using the `map()` function."*
- This sounds more like a **conceptual learning goal** (so you understand *why* it exists) rather than something you absolutely must use in this exact project.

✅ Conclusion

- **No, `map()` is not really needed for this activity** unless you're reading an **analog input** (like a sensor or potentiometer) and scaling it to LED brightness.
- It's likely included so you get familiar with it, even if the main activity doesn't require it.

But we have two digital pins that are not PWMs. Won't we need to use the map() function?

Ah, good catch — you're thinking ahead about the **hardware limitation** 👍

Let's unpack this:

---

1. PWM vs non-PWM pins on Arduino Uno

- `analogWrite()` only works properly on **PWM-capable pins** (on the Uno: 3, 5, 6, 9, 10, 11).
- In your case, pins **8, 9, 10, 11, 12** are used.
  - Only **9, 10, 11** are PWM.
  - Pins **8 and 12** are not PWM, so `analogWrite(...)` on them will just behave like `digitalWrite(...)` → full ON or OFF.

---

2. Does `map()` solve this?

No — `map()` **can't "turn" a non-PWM pin into PWM.**
It only does math to scale values (like 0–1023 → 0–255).
The limitation is in the hardware, not the math.

So even if you used `map()`, pins 8 and 12 would still only do ON/OFF, not smooth brightness levels.

---

3. What this means for your activity

- If your instructor really wants **all five pins** to show brightness control, then either:
  1. The instructions overlooked the PWM limitation, OR
  2. They expect you to realize this and explain that pins 8 and 12 won't fade.
- If it's about **understanding analog signals and** `map()`, then maybe they just want you to apply `map()` on some values for brightness, even if a couple of LEDs don't actually fade.

✅ **Bottom line:**

- `map()` is not needed to *make non-PWM pins work* — it can't fix that.
- But `map()` *might* be expected in your code if the goal is to show you understand scaling values (e.g., mapping a brightness loop 0–100 to 0–255 for PWM).
- The two non-PWM pins will still just act digital ON/OFF unless you use extra hardware (like a software PWM library or external driver).

Is map() function a function that only reads analog signals or analog to digital? If so, is there a way to use it that turns digital pins to usable pwms or analogs?

Great question — you're hitting the core of the confusion many people have when they first see `map()` and `analogWrite()` together 👍

---

1. What `map()` actually is

- `map()` is not an Arduino hardware feature.
- It's just a **math helper function** built into the Arduino core:

```cpp
long map(long x, long in_min, long in_max, long out_min, long out_max) {
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
```

- All it does is **re-scale numbers** from one range to another.
- Example: convert a sensor value `0–1023` to a brightness range `0–255`.

So:

- It doesn't read analog signals.
- It doesn't do analog-to-digital conversion (that's done by the Arduino's ADC hardware when you call `analogRead()`).
- It's just math, nothing more.

---

2. Can `map()` make digital pins behave like PWM or analog?

❌ No.

- PWM capability depends on the **microcontroller hardware timers** inside the ATmega328P (the chip on the Uno).
- If a pin is not tied to a PWM-capable timer (like pins 8 and 12), `map()` can't change that — it only outputs numbers, not hardware signals.

---

3. What you *can* do instead

If you really need PWM on pins like 8 or 12, you have two options:

1. **Rewire your LEDs** to actual PWM pins (simplest).
2. **Software PWM** → You can write a program (or use a library) that rapidly toggles a digital pin ON and OFF in software, simulating PWM. This way, any digital pin can dim an LED.
   - Downside: less smooth, can interfere with timing.

✅ **Summary:**

- `map()` ≠ analog, ≠ PWM. It's just a math function for scaling values.
- It cannot turn a digital pin into PWM.
- Only hardware PWM pins (`3, 5, 6, 9, 10, 11` on Uno) can truly handle brightness control with `analogWrite()`.

So when will the map() function be used?

Perfect — let's clear this up once and for all.

🟢 When `map()` is useful

You'll use `map()` whenever you need to convert values from one scale to another.

Here are the common cases in Arduino projects:

1. **Analog input → PWM output**
   - Example: A potentiometer gives values 0–1023 (`analogRead()` range).
   - LEDs need brightness values 0–255 (`analogWrite()` range).
   - Use `map()` to convert:

```cpp
int sensorValue = analogRead(A0);           // 0–1023
int brightness = map(sensorValue, 0, 1023, 0, 255);
analogWrite(ledPin, brightness);            // 0–255
```

---

2. **Sensor values → meaningful units**
   - Example: A temperature sensor outputs 0–1023.
   - You want °C from 0–100.

```cpp
int rawTemp = analogRead(A1);
int tempC = map(rawTemp, 0, 1023, 0, 100);
```

---

3. **Custom scaling for logic**
   - Example: You want to map a joystick X-axis (0–1023) into motor speeds (–255 to +255).

```cpp
int xValue = analogRead(A2);
int motorSpeed = map(xValue, 0, 1023, -255, 255);
```

🔴 When `map()` is not needed

- If you're just turning LEDs on/off (`digitalWrite`), no need.
- If you already have values in the right range (e.g., 0–255 for PWM), no need.
- If you're trying to make non-PWM pins behave like PWM → `map()` won't help.

✅ In your LED activity

Since your instructions don't mention sensors or inputs, `map()` is probably there as a **learning objective** — to make you aware of how analog inputs can be converted to something usable, like LED brightness.

So:

- You *don't really need* `map()` in the strict sense for just blinking/fading LEDs.
- But you *would* need it if later your instructor asks you to control brightness with, say, a potentiometer or LDR.

> Question: When the brightness exceeds the value of 255 (256, for example), does it go back to values of 1 to 255? For example, if the value is 256, does the ide read it as level 1 brightness?

Nice catch 👌 — that's exactly the kind of detail that trips people up.

Here's how Arduino handles it:

---

1. `analogWrite()` brightness range

- On Arduino Uno, `analogWrite(pin, value)` expects a value from **0 to 255**.
  - `0` = LED completely off.
  - `255` = LED fully on.

---

2. What happens if you go beyond 255?

- Arduino does **not wrap around** (so `256` doesn't become `1`).
- Instead, the value gets **clamped to 255**.
  - Any value ≥ 255 → treated as `255` (full brightness).
  - Any value ≤ 0 → treated as `0` (off).

So:

cpp — Copy code

```cpp
analogWrite(ledPin, 256);   // same as analogWrite(ledPin, 255)
analogWrite(ledPin, 500);   // same as analogWrite(ledPin, 255)
analogWrite(ledPin, -1);    // same as analogWrite(ledPin, 0)
```

✅ Direct answer to your question

No, the IDE **will not loop back to 1–255.**

- `256` = full brightness (same as `255`), not level 1.
- Think of it like a cap: values beyond the limit just "stick" at the max or min.