組込システムⅠ
第４回 課題

提出日 2025/05/15

学籍番号 21T2166D
名前 渡辺 大樹

1　演習 1 - 7 セグメント LED の真理値表

課題: 7 セグメント LED(C-551SRD) の各セグメントの接続状態を示す真理値表を作成しなさい。

以下が 7 セグメント LED の各セグメントの接続状態を示す真理値表である。

数字	A	B	C	D	E	F	G
0	H	H	H	H	H	H	L
1	L	H	H	L	L	L	L
2	H	H	L	H	H	L	H
3	H	H	H	H	L	L	H
4	L	H	H	L	L	H	H
5	H	L	H	H	L	H	H
6	H	L	H	H	H	H	H
7	H	H	H	L	L	L	L
8	H	H	H	H	H	H	H
9	H	H	H	H	L	H	H

2　演習 3 - 7 セグメント LED ルーレットの作成

課題 1: 7 セグメント LED に繰り返しランダムに数字 (0 9) を表示し、スイッチにより任意のタイミングで表示を停止する回路、プログラムを作成しなさい。
課題 2: この回路で、数字の表示、停止を 30 回以上行い、各数字の発生頻度をグラフで報告しなさい。

2.1　課題 1

2.1.1　回路

以下、図 1,2 が 7 セグメント LED に繰り返しランダムに数字 (0 9) を表示し、スイッチにより任意のタイミングで表示を停止する回路, 回路図である。

この回路は、セグメントの接続状態を示す真理値表を元に、7 セグメント LED の各セグメントに接続された LED を点灯させるための回路である。また、スイッチを押すことで、7 セグメント LED に表示される数字を停止させることができる。

Raspberry Pi Pico の GPIO ピンを使用して、7 セグメント LED の各セグメントを制御している。接続先はそれぞれ

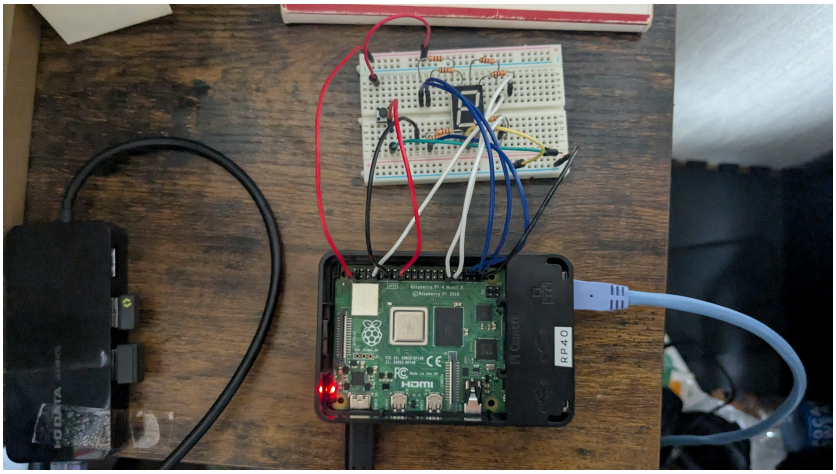

図 1　7 セグメント LED ルーレット

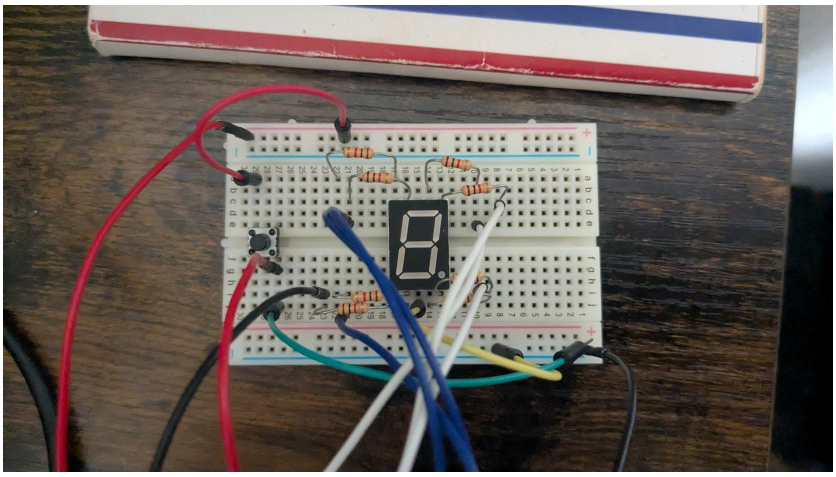

図 2　7 セグメント LED ルーレット回路図

- A: GPIO 4
- B: GPIO 5
- C: GPIO 6
- D: GPIO 16
- E: GPIO 17
- F: GPIO 20
- G: GPIO 21

となっている。またタクトスイッチは GPIO 22 に接続されている。

2.1.2　使用部品

- Raspberry Pi 4 Model B

- 7 セグメント LED(C-551SRD)
- タクトスイッチ
- 抵抗 (1k Ω) * 8
- ブレッドボード
- ジャンパーワイヤー

2.1.3 プログラム

以下ソースコード 1 が 7 セグメント LED に繰り返しランダムに数字 (0 9) を表示し、スイッチにより任意のタイミングで表示を停止するプログラムである。

このコードは 7 セグメント LED に 0 から 9 までの数字をランダムに表示し、スイッチが押されたときに表示を停止する機能を実装している。主な特徴は以下の通りである：

- プログラムは `RPi.GPIO` ライブラリを用いて Raspberry Pi の各種 GPIO ピンを制御している
- 7 セグメント LED の各セグメント（A〜G）に対応する GPIO ピンを配列として定義し、効率的な制御を実現している
- 数字 0〜9 の表示パターンを 2 次元配列として定義し、各セグメントの点灯・消灯状態を管理している
- スイッチ入力検出時には表示中の数字をカウントし、結果を保存する機能を実装している
- マルチスレッド処理により、LED 表示制御とスイッチ入力の並列監視を実現している
- 表示中の数字をランダムに変更するため、`random` ライブラリを使用している

特に、カウント機能をつけたことで、容易に課題 2 であるカウントの実施が容易となっている。また shell の input を監視することで stop を入力すると、ルーレットが止まるようにしている。

2.2 課題 2

課題 2: この回路で、数字の表示、停止を 30 回以上行い、各数字の発生頻度をグラフで報告しなさい。

以下に課題 2 の結果を示す。

スイッチ押下回数: 45
0: 4 回
1: 9 回

2: 3 回
3: 3 回
4: 2 回
5: 4 回
6: 5 回
7: 4 回
8: 6 回
9: 5 回

グラフで示すと以下の図 3 のようになる。

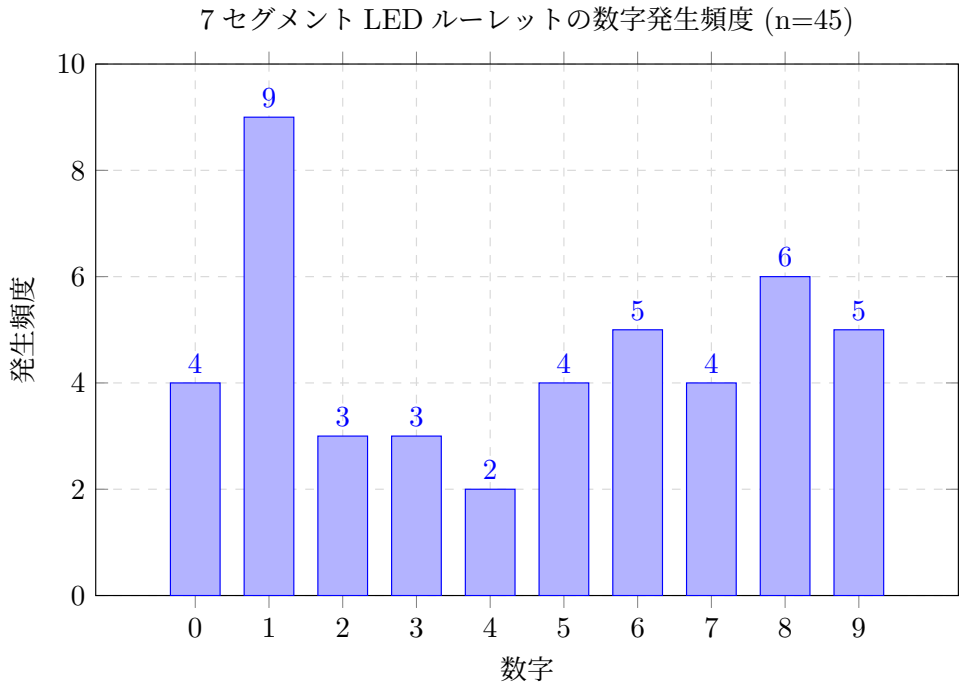

図 3　7 セグメント LED ルーレットの発生頻度分布

2.3　考察

7 セグメント LED ルーレットの数字発生頻度を調査した結果、数字 1 が最も多く出現し、次いで数字 8 が多く出現した。逆に数字 4 が最も少なく出現した。
また、全体的に数字の出現頻度は均等ではなく、特定の数字が多く出現する傾向が見られた。
今後の改善点としては、より均等な出現頻度を実現するために、乱数生成アルゴリズムの見直しや、スイッチ押下時の処理を改良することが考えられる。

3　問い - A/D 変換に関する「量子化誤差」について

問い: A/D 変換に関する「量子化誤差」について説明しなさい。

量子化誤差とは、アナログ信号をデジタル信号に変換する際に生じる誤差のことを指す。
A/D 変換器は、連続的なアナログ信号を離散的なデジタル値に変換するが、この過程で信号の値を近似するため、元の信号と変換後の信号との間に誤差が生じる。
この誤差は、量子化ステップと呼ばれる最小の変化単位に依存し、量子化ステップが小さいほど誤差は小さくなる。
量子化誤差は、信号の精度や品質に影響を与えるため、特に音声や画像処理などの分野で重要な要素となる。
量子化誤差を最小限に抑えるためには、A/D 変換器の分解能を高くすることや、適切なフィルタリング技術を使用することが重要である。

ソースコード 1　exam4.py

```
1  import RPi.GPIO as GPIO # type: ignore
2  import time
3  import random
4  import threading
5
6  SEGMENTS = {
7      'A': 4,
8      'B': 5,
9      'C': 6,
10     'D': 16,
11     'E': 17,
12     'F': 20,
13     'G': 21,
14 }
15
16 SWITCH_PIN = 22
17
18 DIGIT_TO_SEGMENTS = {
19     0: ['A', 'B', 'C', 'D', 'E', 'F'],
20     1: ['B', 'C'],
21     2: ['A', 'B', 'D', 'E', 'G'],
22     3: ['A', 'B', 'C', 'D', 'G'],
23     4: ['B', 'C', 'F', 'G'],
```

```
    5: ['A', 'C', 'D', 'F', 'G'],
    6: ['A', 'C', 'D', 'E', 'F', 'G'],
    7: ['A', 'B', 'C'],
    8: ['A', 'B', 'C', 'D', 'E', 'F', 'G'],
    9: ['A', 'B', 'C', 'D', 'F', 'G'],
}

digit_counts = {i: 0 for i in range(10)}
total_switch_presses = 0
stop_flag = False

def setup():
    GPIO.setmode(GPIO.BCM)
    for pin in SEGMENTS.values():
        GPIO.setup(pin, GPIO.OUT)
        GPIO.output(pin, GPIO.LOW)
    GPIO.setup(SWITCH_PIN, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

def display_digit(digit):
    segments_on = DIGIT_TO_SEGMENTS[digit]
    for seg, pin in SEGMENTS.items():
        GPIO.output(pin, GPIO.HIGH if seg in segments_on else GPIO.LOW)

def input_thread():
    global stop_flag
    while True:
        user_input = input()
        if user_input.strip().lower() == 'stop':
            stop_flag = True
            break

def main():
    global total_switch_presses
    setup()
    threading.Thread(target=input_thread, daemon=True).start()

    last_digit = 0
    switch_was_pressed = False

    try:
        while not stop_flag:
            digit = random.randint(0, 9)
```

```
            display_digit(digit)
            last_digit = digit

            if GPIO.input(SWITCH_PIN) == GPIO.HIGH:
                if not switch_was_pressed:
                    switch_was_pressed = True
                    total_switch_presses += 1
                    digit_counts[last_digit] += 1
                    display_digit(last_digit)
                while GPIO.input(SWITCH_PIN) == GPIO.HIGH:
                    time.sleep(0.05)
                continue
            else:
                switch_was_pressed = False

            time.sleep(0.05)
    finally:
        GPIO.cleanup()
        print(f"\n スイッチ押下回数:␣{total_switch_presses}")
        for digit in range(10):
            print(f"{digit}:␣{digit_counts[digit]}回")

if __name__ == '__main__':
    main()
```