組込システムⅠ
第８回 課題

提出日 2025/06/12

学籍番号 21T2166D
名前 渡辺 大樹

1　課題

本課題では、Raspberry Pi と 4 桁 7 セグメント LED、BCD-to-7 セグメントデコーダ IC（74HC4511）を用いて、デジタル時計およびカレンダー表示システムを実装する。具体的には、演習（2）として以下の機能を実現する：

1. Raspberry Pi のシステム時刻を取得し、4 桁 7 セグメント LED に表示する。
2. 表示モードは「時・分（HHMM）」と「月・日（MMDD）」の 2 種類とする。
3. タクトスイッチを押すことで、表示モードを切り替える。
4. 時刻および日付の十の位が 0 の場合は、その桁をブランク（非表示）にする。例えば、8 時 5 分は「 8 05」、3 月 5 日は「 3 05」のように表示する。

本課題を通じて、7 セグメント LED のダイナミック点灯（マルチプレキシング）の原理、BCD デコーダ IC の利用方法、イベント駆動によるスイッチ入力処理、および Python の datetime モジュールを用いた時刻情報の扱について理解を深めることを目的とする。

2　使用部品

本演習で使用した主な部品は以下の通りである。

- Raspberry Pi 4B
- 4 桁 7 セグメント LED（カソードコモン、例：OptoSupply OSL40562-LR）
- BCD-to-7 セグメントデコーダ IC（74HC4511）
- タクトスイッチ（1 個）
- 抵抗：1kΩ（7 個、セグメント電流制限用、74HC4511 の出力と LED セグメント間。ただし、スライドの回路図には 1kΩ とあるが、74HC4511 の出力電流能力と LED の仕様に応じて適切な値を選定する。今回はスライドの指示に従う。）
- ブレッドボード
- ジャンパー線

3　回路の説明

本システムでは、Raspberry Pi の GPIO ピンを使用して、74HC4511 デコーダ IC、4 桁 7 セグメント LED、およびタクトスイッチを制御する。

使用した Raspberry Pi の GPIO ピン（BCM 番号と物理ピン番号）：

- BCD データ出力 (74HC4511 へ):
  - GPIO20 (Pin 38) - D0
  - GPIO21 (Pin 40) - D1
  - GPIO22 (Pin 15) - D2
  - GPIO23 (Pin 16) - D3
- 7 セグメント LED 桁選択 (カソード制御):
  - GPIO24 (Pin 18) - DIG1
  - GPIO25 (Pin 22) - DIG2
  - GPIO26 (Pin 37) - DIG3
  - GPIO27 (Pin 13) - DIG4
- スイッチ入力:
  - GPIO16 (Pin 36) - Switch

- 電源:
  - +3.3V (Pin 1 or Pin 17)
  - GND (Pin 39 or others)

回路構成：

- Raspberry Pi の GPIO20-GPIO23 を 74HC4511 の BCD 入力 D0-D3 に接続する。
- 74HC4511 のセグメント出力 a-g を、それぞれ 1kΩ の電流制限抵抗を介して 4 桁 7 セグメント LED の対応するセグメント入力に接続する。
- 74HC4511 の制御ピンは、LT (Lamp Test) と BL (Blanking) を +3.3V に、LE (Latch Enable) を GND に接続し、トランスペアレントモードで動作させる。VCC は +3.3V、GND は GND に接続する。
- Raspberry Pi の GPIO24-GPIO27 を 4 桁 7 セグメント LED の各桁のコモンカソード（DIG1-DIG4）に接続する。これにより、特定の桁を LOW にすることでその桁を点灯させる。
- タクトスイッチの一端を Raspberry Pi の GPIO16 に、もう一端を +3.3V に接続する。GPIO16 は内部プルダウン抵抗を有効にして使用する。

4 回路図

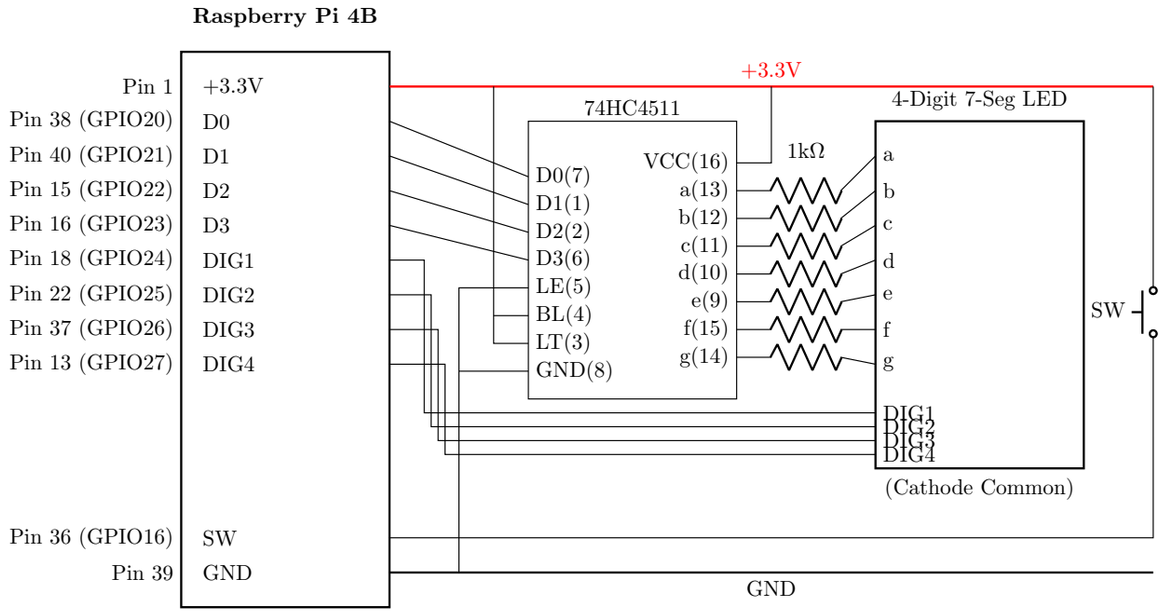

図 1　演習（2）の回路図

5 アルゴリズムの説明

本システムの制御は Python スクリプト（‘exam8-2.py‘）によって行われる。アルゴリズムの主要な部分は以下の通りである。

1. **初期設定 (‘setup‘関数):**
   - GPIO のモードを BCM に設定。
   - BCD 出力ピン (D0-D3) を出力モードに設定。
   - 桁選択ピン (DIG1-DIG4) を出力モードに設定し、初期状態を HIGH（全桁非表示）にする。カソードコモンなので LOW で点灯。
   - スイッチ入力ピン (PIN_SWITCH) を入力モードに設定し、内部プルダウン抵抗を有効にする。

- スイッチ入力ピンに立ち上がりエッジ検出のイベントを設定し、コールバック関数 ‘switch_callback‘ を登録する。チャタリング防止のため、‘bouncetime‘を 300ms に設定。

2. **スイッチ処理 (‘switch_callback‘関数):**
   - スイッチが押されるたびにグローバル変数 ‘display_mode‘ の値を ’time’ と ’date’ の間でトグルする。
3. **BCD データ送信 (‘send_to_bcd‘関数):**
   - 表示したい数字（0-9）を引数として受け取る。
   - 事前に定義された BCD コードの辞書 ‘BCD_CODES‘ を参照し、対応する 4 ビットの BCD データを取得する。
   - 0-9 以外の数字（特に、先頭の 0 を消すために使用する 15）が指定された場合は、全ビットが 1 のコード（通常はブランク表示）を出力する。
   - 取得した BCD データを GPIO ピン (D0-D3) に出力する。
4. **表示ループ (‘display_loop‘関数):**
   - 無限ループで以下の処理を繰り返す（ダイナミック点灯）。
   - 現在の日時を ‘datetime.datetime.now()‘ で取得する。
   - ‘display_mode‘ の値に応じて、表示する 4 つの数字（時・分または月・日）をリスト ‘display_digits‘ に格納する。
   - 時・分表示の場合：‘[時 (十の位), 時 (一の位), 分 (十の位), 分 (一の位)]‘
   - 月・日表示の場合：‘[月 (十の位), 月 (一の位), 日 (十の位), 日 (一の位)]‘
   - 各ペア（時、分、月、日）の十の位が 0 の場合、その桁をブランク表示にするため、対応する ‘display_digits‘ の要素を 15 に置き換える。
   - 4 つの桁を順番に高速に切り替えて点灯させる（ダイナミック点灯）。
     （a）全ての桁選択ピンを HIGH にして全桁を消灯する。
     （b）表示する桁の数字に対応する BCD データを ‘send_to_bcd‘ 関数で 74HC4511 に送信する。
     （c）対応する桁選択ピンを LOW にしてその桁を点灯させる。
     （d）短時間 (0.002 秒) 待機する。この遅延がダイナミック点灯の周期を決定する。
5. **メイン処理 (‘main‘関数):**
   - ‘setup‘関数を呼び出して初期化を行う。
   - ‘display_loop‘関数を呼び出して表示を開始する。
   - ‘KeyboardInterrupt‘ (Ctrl+C) を検出するとプログラムを終了し、‘GPIO.cleanup()‘ で GPIO 設定を解放する。

このアルゴリズムにより、少ない GPIO ピンで 4 桁の数字を表示し、スイッチ入力で表示内容を切り替えるシステムが実現される。

6 結果

本課題で実装したシステムは、設計通りに動作することを確認した。
また実験した際の Raspberry Pi の本体時刻は ‘print(datetime.datetime.now())‘より ‘2025-05-09 18:22:25.162019‘と確認できた

1. 電源投入後、4 桁 7 セグメント LED には現在の時刻が「HHMM」形式で「18 21」と表示された。
2. タクトスイッチを押すと、表示が現在の月日に「MMDD」形式で切り替わり「 5 9」と表示された。
3. 再度タクトスイッチを押すと、表示は時刻表示に戻った。この切り替え動作はスイッチを押すたびに繰り返された。
4. ダイナミック点灯により、4 つの桁が連続的に点灯しているように見え、チラツキも許容範囲内であった。‘time.sleep(0.002)‘ の設定により、各桁の点灯時間は 2ms となり、全体のフレームレートは約 125Hz $(1/(4 \times 0.002\text{s}))$ となるため、人間の目にはほぼ連続点灯として認識された。

5. スイッチのチャタリングは ‘bouncetime=300ms‘ の設定により効果的に防止され、一度の押下で確実に表示モードが切り替わった。

以下に、実際にブレッドボード上に組んだ回路の写真を示す（図 2）。写真では、Raspberry Pi、ブレッドボード、7

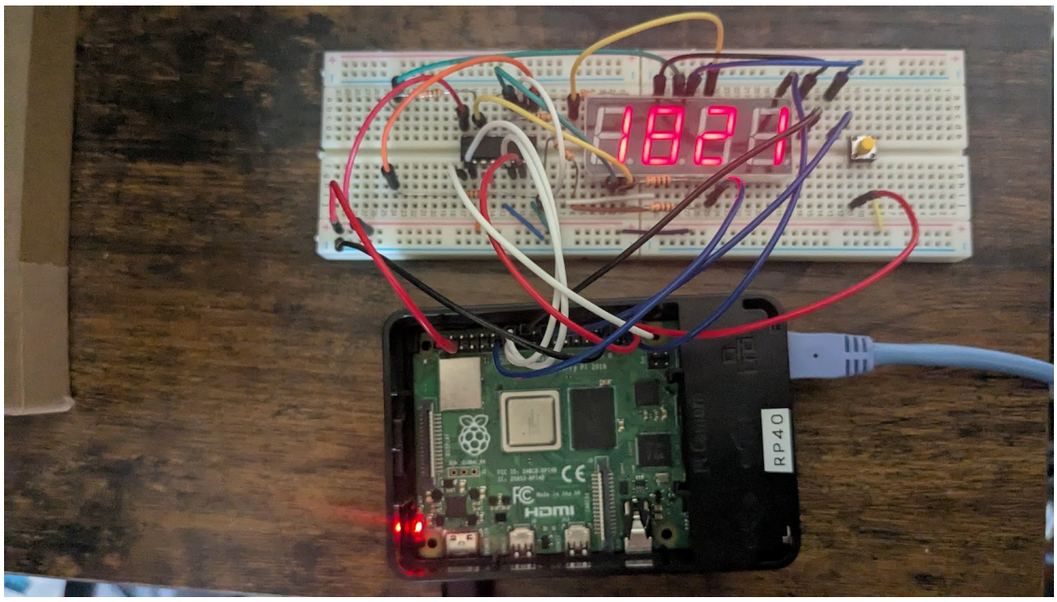

図 2　実装した回路の写真

セグメント LED、74HC4511 IC、タクトスイッチ、および配線が確認できる。LED には時刻または日付が意図した通りに表示された。

7　考察

本演習を通じて、以下の点について理解を深めることができた。

- **ダイナミック点灯の有効性：** 4 桁の 7 セグメント LED を駆動するために、セグメント制御用（7 本、BCD デコーダ経由なら 4 本）と桁選択用（4 本）の GPIO ピンのみで済むため、ピン数の限られたマイクロコントローラにおいて非常に有効な技術である。本演習では BCD デコーダ IC（74HC4511）を使用したため、BCD 入力 4 ピン＋桁選択 4 ピンの計 8 ピンで 4 桁の数字を表示できた。ソフトウェアによる高速な切り替えにより、人間の目には全桁が同時に点灯しているように見える残像効果を利用している点を実感した。
- **BCD デコーダ IC の利便性：** 74HC4511 のような BCD-to-7 セグメントデコーダを使用することで、マイクロコントローラ側は表示したい数字の BCD コードを出力するだけで済み、7 セグメントの各セグメントを直接制御する複雑なロジックをソフトウェアで実装する必要がなくなる。これにより、CPU 負荷の軽減とプログラムの簡略化が図れる。
- **イベント駆動処理：** スイッチ入力の処理に ‘GPIO.add_event_detect‘ を用いたイベント駆動方式を採用することで、CPU が常にスイッチの状態をポーリングする必要がなくなり、リソースを効率的に使用できる。コールバック関数により、スイッチが押された瞬間にのみ対応する処理が実行されるため、応答性も向上する。
- **datetime モジュールの活用：** Python の ‘datetime‘ モジュールを使用することで、現在時刻や日付の取得、およびそれらの要素（時、分、月、日）へのアクセスが容易に行えることを確認した。
- **表示の工夫：** 時刻や日付の十の位が 0 の場合にブランク表示にする処理は、ユーザーインターフェースの観点から視認性を高める効果がある。74HC4511 では BCD コード 10〜15 がブランク表示となる特性を利用してこれを実現した。
- **回路設計の注意点：** 7 セグメント LED の各セグメントには適切な電流制限抵抗を接続することが重要である。また、74HC4511 の制御ピン（LE, BL, LT）の処理（プルアップ/プルダウン）も IC の仕様に従って正しく

行う必要がある。今回は LE を GND に接続してトランスペアレントモードとした。

今後の改善点としては、表示のチラツキをさらに低減するためのリフレッシュレートの最適化や、より多くの情報を表示するための工夫（例えばドットマトリクス LED の使用）などが考えられる。

8 問いの解答

問い：4 桁 LED がアノードコモンであった場合、今回の演習はどの様な構成とすれば良いか。

解答：4 桁 7 セグメント LED がアノードコモンであった場合、回路構成と制御方法にいくつかの変更が必要となる。主な変更点は以下の通りである。

1. **桁選択（コモンアノード制御）：**
   - アノードコモンの場合、各桁の共通アノード端子に +3.3V（または適切な駆動電圧）を供給することでその桁がアクティブになる。Raspberry Pi の GPIO ピンで直接駆動する場合、桁選択ピン（DIG1〜DIG4）は HIGH を出力することで対応する桁を選択（点灯準備）し、LOW で非選択とする。
   - Python コードでは、桁選択部分のロジックが逆になる。初期状態で全桁選択ピンを LOW（非アクティブ）にし、点灯させたい桁のピンを HIGH にする。

```
# 初期化時（setup 関数内）
# GPIO.output(pin, GPIO.HIGH) を GPIO.output(pin, GPIO.LOW) に変更
for pin in DIGIT_PINS:
    GPIO.setup(pin, GPIO.OUT)
    GPIO.output(pin, GPIO.LOW) # 全桁を初期状態で非表示


# 表示ループ内（display_loop 関数内）
# GPIO.output(DIGIT_PINS[i], GPIO.LOW) を GPIO.output(DIGIT_PINS[i], GPIO.HIGH) に変更
GPIO.output(DIGIT_PINS[i], GPIO.HIGH) # 選択した桁を点灯
# 消灯時は GPIO.output(pin, GPIO.HIGH) を GPIO.output(pin, GPIO.LOW) に変更
for pin in DIGIT_PINS:
    GPIO.output(pin, GPIO.LOW) # 全桁を非表示
```

   - GPIO の出力電流能力を超える場合は、PNP トランジスタや専用のドライバ IC（例：ULN2003A の逆の使い方や TD62783A など）を介してアノードを駆動する必要がある。
2. **セグメント制御：**
   - アノードコモンの場合、各セグメント（a〜g）は対応するピンを LOW にすることで点灯する。
   - 使用している 74HC4511 は、BCD 入力に応じてアクティブ HIGH（HIGH でセグメント点灯）の信号をセグメント出力ピンに出力する。これはカソードコモン LED に適している。
   - アノードコモン LED で 74HC4511 をそのまま使用する場合、74HC4511 の各セグメント出力と LED のセグメント入力の間に NPN トランジスタやインバータ IC（例：74HC04）を挟み、信号を反転させる必要がある。
   - あるいは、アノードコモン用の BCD-to-7 セグメントデコーダ IC（例：74LS48 など、アクティブ LOW 出力を持つもの）に変更する。この場合、74LS48 は TTL レベルなので、Raspberry Pi の 3.3V ロジックとのレベル変換に注意が必要な場合がある。CD4543B などもアノードコモンに対応できる設定がある。
   - 電流制限抵抗は引き続き各セグメントに必要となる。
3. **74HC4511 の制御ピン（LE, BL, LT）と電源：**
   - これらの制御は LED の種類（アノードコモン／カソードコモン）に直接依存しないため、変更は不要。VCC, GND の接続も同様。

まとめると、アノードコモン LED を使用する場合は、主に桁選択の駆動方法（GPIO 出力の論理反転、場合によってはドライバ追加）と、セグメント信号の論理反転（デコーダ IC の変更または反転回路の追加）が必要となる。ソフトウェア側では、桁選択の GPIO 出力論理を反転させる修正が主となる。

9　付録：ソースコード

ソースコード 1　exam8-2.py

```
import RPi.GPIO as GPIO
import datetime
import time

PIN_D0 = 20 # 8
PIN_D1 = 21 # 4
PIN_D2 = 22 # 2
PIN_D3 = 23 # 1
BCD_PINS = [PIN_D0, PIN_D1, PIN_D2, PIN_D3]

PIN_DIG1 = 24
PIN_DIG2 = 25
PIN_DIG3 = 26
PIN_DIG4 = 27
DIGIT_PINS = [PIN_DIG1, PIN_DIG2, PIN_DIG3, PIN_DIG4]

PIN_SWITCH = 16

display_mode = 'time'

BCD_CODES = {
    0: (0,0,0,0), 1: (1,0,0,0), 2: (0,1,0,0), 3: (1,1,0,0), 4: (0,0,1,0),
    5: (1,0,1,0), 6: (0,1,1,0), 7: (1,1,1,0), 8: (0,0,0,1), 9: (1,0,0,1),
    15: (1,1,1,1)
}

def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
    for pin in BCD_PINS:
        GPIO.setup(pin, GPIO.OUT)
    for pin in DIGIT_PINS:
        GPIO.setup(pin, GPIO.OUT)
        GPIO.output(pin, GPIO.HIGH) # 全桁を初期状態で非表示
    GPIO.setup(PIN_SWITCH, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
    GPIO.add_event_detect(PIN_SWITCH, GPIO.RISING, callback=switch_callback, bouncetime=300)

def switch_callback(channel):
    global display_mode
    if display_mode == 'time':
        display_mode = 'date'
    else:
        display_mode = 'time'

```

```
45 def send_to_bcd(number):
46     code = BCD_CODES.get(number, BCD_CODES[15])
47     for i in range(4):
48         GPIO.output(BCD_PINS[i], code[i])
49 
50 def display_loop():
51     while True:
52         now = datetime.datetime.now()
53 
54         if display_mode == 'time':
55             h1 = now.hour // 10
56             h2 = now.hour % 10
57             m1 = now.minute // 10
58             m2 = now.minute % 10
59             display_digits = [h1, h2, m1, m2]
60         else:
61             mo1 = now.month // 10
62             mo2 = now.month % 10
63             d1 = now.day // 10
64             d2 = now.day % 10
65             display_digits = [mo1, mo2, d1, d2]
66 
67         if display_digits[0] == 0:
68             display_digits[0] = 15
69         if display_digits[2] == 0:
70             display_digits[2] = 15
71 
72         for i in range(4):
73             for pin in DIGIT_PINS:
74                 GPIO.output(pin, GPIO.HIGH)
75 
76             send_to_bcd(display_digits[i])
77 
78             GPIO.output(DIGIT_PINS[i], GPIO.LOW)
79 
80             time.sleep(0.002)
81 
82 def main():
83     try:
84         setup()
85         display_loop()
86     except KeyboardInterrupt:
87         pass
88     finally:
89         GPIO.cleanup()
90 
91 if __name__ == '__main__':
92     main()
```