組込システムⅠ
第７回 課題

提出日 2025/06/05

学籍番号 21T2166D
名前 渡辺 大樹

1　課題

本課題では、イベント検出と電圧比較回路を用いた組込システムの実装を行った。具体的には以下の 2 つの演習を実施した：

1. 演習（1）：RPi.GPIO ライブラリを用いたスイッチ入力のイベント検出システムの実装
2. 演習（3）：CDS 素子と電圧比較回路（LM339）を用いた光センサーシステムの実装

これらの演習を通じて、ポーリング方式とイベント検出方式の違い、および電圧比較回路の動作原理について理解を深めることを目的とした。

2　使用部品

2.1　演習（1）で使用した部品

- Raspberry Pi 4B
- タクトスイッチ（1 個）
- ブレッドボード
- ジャンパー線

2.2　演習（3）で使用した部品

- Raspberry Pi 4B
- CDS 光センサー（1 個）
- LM339 電圧比較器 IC（1 個）
- LED（1 個）
- タクトスイッチ（1 個）
- 抵抗：6.8k Ω（1 個）、1k Ω（1 個）
- 可変抵抗：10k Ω（1 個）
- ブレッドボード
- ジャンパー線

3　回路の説明

3.1　演習（1）の回路構成

演習（1）では、GPIO21 番ピンにタクトスイッチを接続した。スイッチの一端は GND、もう一端は GPIO21 に接続し、内部プルアップ抵抗を有効にしてスイッチ入力を検出した。

使用した Raspberry Pi の端子：

- GPIO21（Pin 40）：スイッチ入力
- GND（Pin 39）：グランド接続

3.2　演習（3）の回路構成

演習（3）では、より複雑な回路を実装した。CDS 素子と 6.8k Ω抵抗による分圧回路の出力を LM339 の非反転入力（IN+）に、10k Ω可変抵抗による分圧回路の出力を反転入力（IN-）に接続した。LM339 の出力はオープンコレ

クタタイプのため、プルアップ抵抗を介して GPIO20 に接続した。

使用した Raspberry Pi の端子：

- GPIO16（Pin 36）：LED 制御出力
- GPIO20（Pin 38）：比較器出力入力
- GPIO21（Pin 40）：スイッチ入力
- +3.3V（Pin 1）：電源供給
- GND（Pin 39）：グランド接続

4　回路図

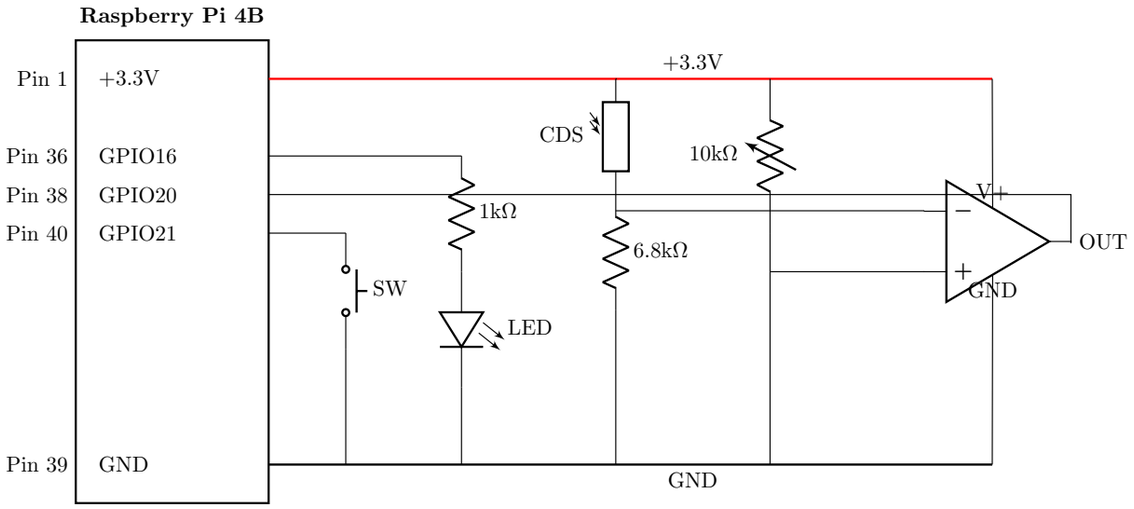

図 1　演習（3）の回路図

5　アルゴリズムの説明

5.1　演習（1）のアルゴリズム

演習（1）では、イベント検出を用いたスイッチ入力処理を実装した。アルゴリズムの流れは以下の通りである：

1. GPIO21 を入力ピンとして設定し、内部プルアップ抵抗を有効化
2. 立ち下がりエッジでイベントを検出するよう設定
3. コールバック関数 `checkSW` をイベント発生時に呼び出すよう登録
4. メインループでフラグ `s` を監視し、スイッチの状態を表示
5. チャタリング防止のため 200ms の bounce time を設定

この方式により、CPU がポーリングを行う必要がなく、効率的なスイッチ入力検出が可能となった。

5.2　演習（3）のアルゴリズム

演習（3）では、2 つのイベント検出を同時に処理するシステムを実装した：

1. GPIO 設定：
   - GPIO16 を LED 制御用出力として設定
   - GPIO20 を CDS 比較器出力用入力として設定（プルアップ有効）
   - GPIO21 をスイッチ用入力として設定（プルアップ有効）

2. イベント検出設定：
   - スイッチ（GPIO21）の立ち下がりエッジで `sw_callback` を呼び出し
   - CDS 比較器（GPIO20）の立ち下がりエッジで `cds_callback` を呼び出し
3. 処理ロジック：
   - 暗くなると CDS 比較器が LOW を出力し、LED を点灯
   - スイッチが押されると LED を消灯
   - グローバル変数 `led_on` で LED 状態を管理

6 結果

6.1 演習（1）の結果

演習（1）では、スイッチを押すたびに「SW on」、スイッチを離すと「SW off」がコンソールに 1 秒間隔で表示された。イベント検出により、スイッチの状態変化を即座に検知できることが確認された。bounceTime=200ms の設定により、チャタリングによる誤検出も効果的に防止された。

従来のポーリング方式と比較して、CPU 使用率の削減とリアルタイム性の向上が実現された。

6.2 演習（3）の結果

演習（3）では、以下の動作が正常に確認された：

1. CDS 素子に光が当たっている状態では、比較器出力が HIGH となり LED は消灯状態を維持
2. CDS 素子を手で覆い暗くすると、比較器出力が LOW となり「暗さ検知：LED 点灯」が表示され LED が点灯
3. LED 点灯中にスイッチを押すと、「SW 押下：LED 消灯」が表示され LED が消灯
4. 可変抵抗を調整することで、光の閾値を変更可能であることを確認

実装した回路では、オープンコレクタ出力に対するプルアップ抵抗の必要性と、適切なエッジ検出の重要性が実証された。

7 考察

7.1 イベント検出方式の優位性

今回の実験を通じて、ポーリング方式とイベント検出方式の違いを実感できた。イベント検出方式では以下の利点が確認された：

- CPU 使用率の削減：常時ポーリングが不要
- リアルタイム性の向上：状態変化を即座に検知
- 消費電力の削減：待機時の処理負荷軽減

特に組込システムにおいては、限られたリソースを効率的に活用する観点から、イベント駆動型の設計が重要であることが理解できた。

7.2 電圧比較回路の応用

LM339 を用いた電圧比較回路は、アナログ信号をデジタル信号に変換する重要な役割を果たした。CDS 素子による光強度の変化を電圧変化として捉え、閾値との比較によりデジタル判定を行うことで、マイコンでの処理が可能となった。

可変抵抗による閾値調整機能により、環境条件に応じた柔軟な設定が可能であることも確認できた。

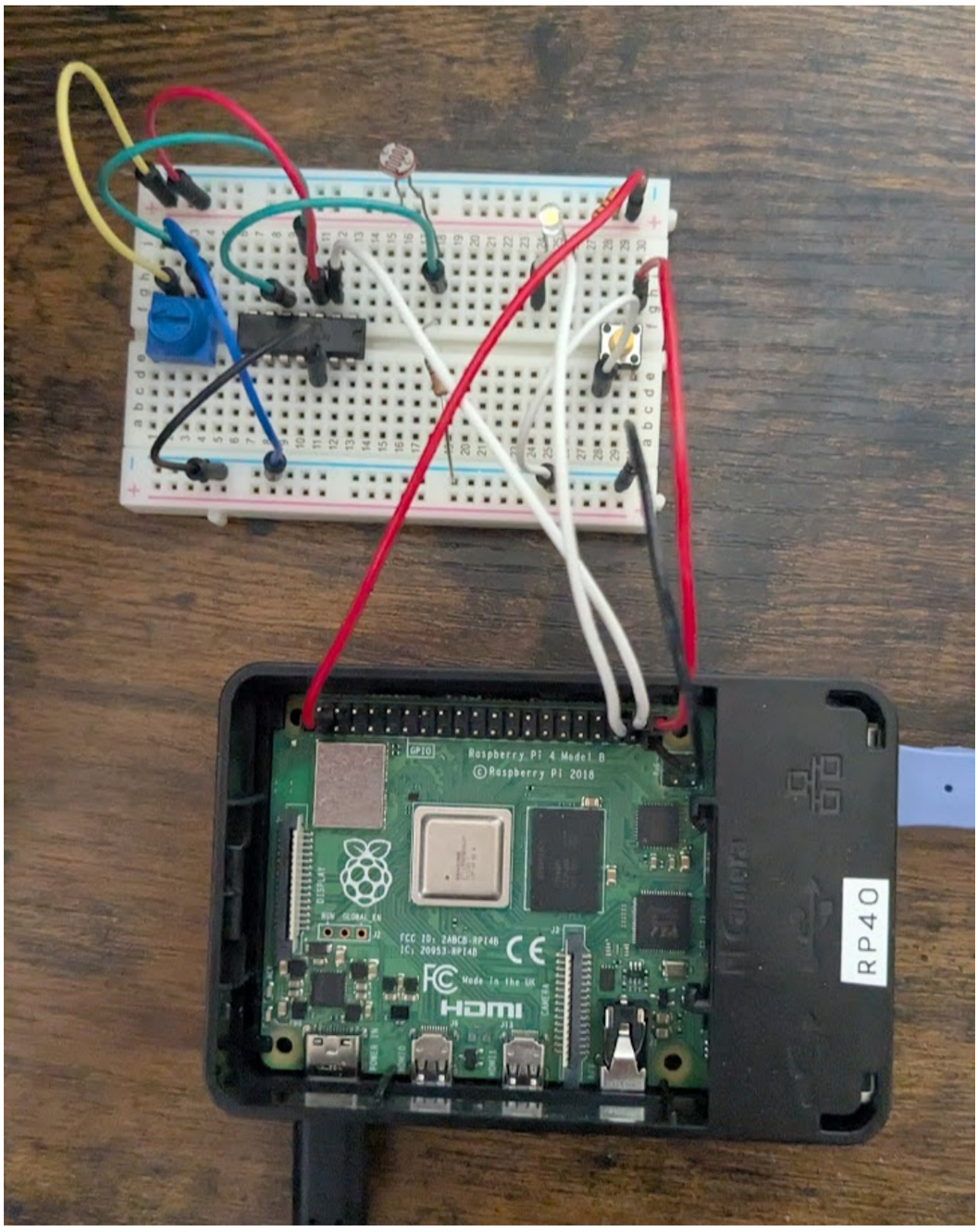

図 2　実装した回路の写真

7.3　オープンコレクタ出力の特性

LM339 のオープンコレクタ出力特性により、プルアップ抵抗が必須であることを実験を通じて学んだ。この構成により、複数の出力をワイヤード OR 接続することも可能であり、システム拡張性の観点でも有用な特性である。

8　問いの解答

問い：オープンコレクタとはどの様なものか。

解答：オープンコレクタとは、トランジスタのコレクタが出力端子となっており、プルアップ抵抗を介して電源に接続する必要がある出力形式である。

具体的な動作は以下の通りである：

- **LOW レベル出力時：**内部のトランジスタが ON 状態となり、出力端子が GND（0V）に接続される
- **HIGH レベル出力時：**内部のトランジスタが OFF 状態となり、出力端子は外部のプルアップ抵抗を介して電源電圧まで引き上げられる

この構成の利点として、以下が挙げられる：

1. 複数の出力を並列接続（ワイヤード OR）可能
2. 異なる電源電圧レベルとのインターフェースが容易
3. 消費電力の削減が可能

ただし、プルアップ抵抗が必須であり、これを忘れると正常な動作が得られないため注意が必要である。

9 付録：ソースコード

9.1 演習（1）のソースコード

ソースコード 1 exam7-1.py

```
1  import RPi.GPIO as GPIO
2  import time
3
4  GPIO.setmode(GPIO.BCM)
5  SW = 21
6  GPIO.setup(SW, GPIO.IN, pull_up_down=GPIO.PUD_UP)
7
8  s = 0
9
10 def checkSW(pin):
11     global s
12     s = 1
13
14 GPIO.add_event_detect(SW, GPIO.FALLING, callback=checkSW, bouncetime=200)
15
16 try:
17     while True:
18         if s == 1:
19             print("SW on")
20             s = 0
21         else:
22             print("SW off")
23         time.sleep(1)
24
25 except KeyboardInterrupt:
26     pass
27 finally:
28     GPIO.remove_event_detect(sw)
29     GPIO.cleanup()
```

9.2 演習（3）のソースコード

ソースコード 2 exam7-3.py

```
1  import RPi.GPIO as GPIO
2  import time
3
4  SW = 21 # スイッチのGPIO ピン
5  CDS = 20 # CDS 出力（比較器出力）の GPIO ピン
6  LED = 16 # LED の GPIO ピン
7
8  GPIO.setmode(GPIO.BCM)
9  GPIO.setup(SW, GPIO.IN, pull_up_down=GPIO.PUD_UP)
10 GPIO.setup(CDS, GPIO.IN, pull_up_down=GPIO.PUD_UP)
11 GPIO.setup(LED, GPIO.OUT)
12
```

```
led_on = False

def sw_callback(channel):
    global led_on
    if led_on:
        GPIO.output(LED, GPIO.HIGH)
        led_on = False
        print("SW 押下：LED 消灯")

def cds_callback(channel):
    global led_on
    if GPIO.input(CDS) == GPIO.HIGH: # 暗いときにLOW になる設定
        GPIO.output(LED, GPIO.LOW)
        led_on = True
        print("暗さ検知:LED 点灯")

GPIO.add_event_detect(SW, GPIO.FALLING, callback=sw_callback, bouncetime=200)
GPIO.add_event_detect(CDS, GPIO.FALLING, callback=cds_callback, bouncetime=200)

try:
    while True:
        time.sleep(1)
except KeyboardInterrupt:
    pass
finally:
    GPIO.cleanup()
```