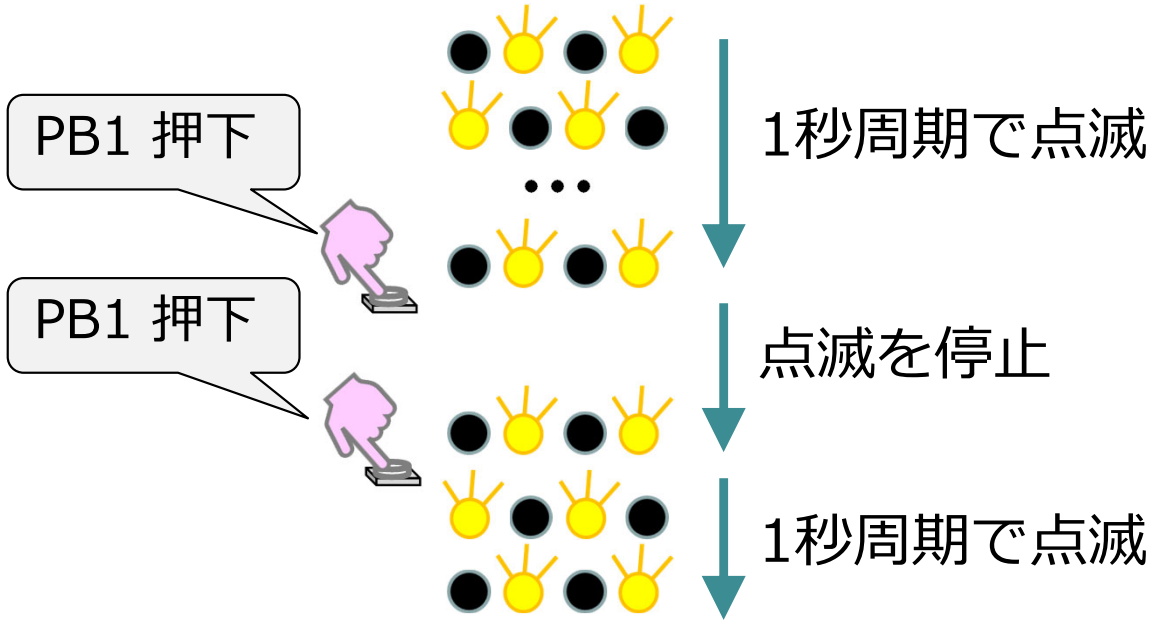
例題9　状態変数

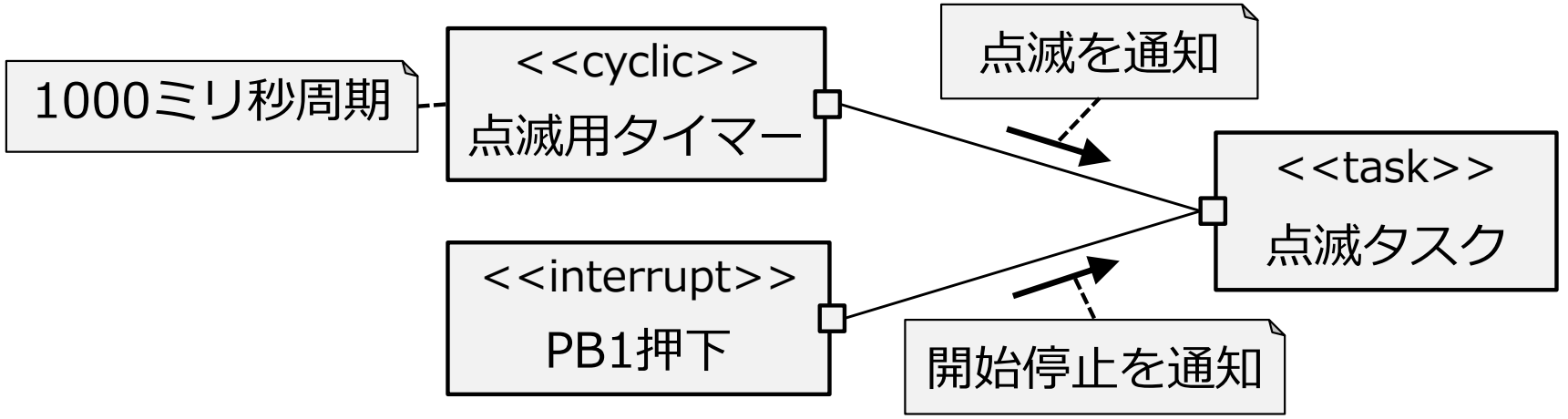
- プッシュボタン（PB1）の押下によって LED の点滅を停止したり、継続したりする
  - プログラムが始まると LED を 1 秒周期で点滅する
  - プッシュボタン（PB1）が押下されると 点滅を停止する
  - 再びプッシュボタン（PB1）が押下されると 点滅を再開する

  以下繰り返す



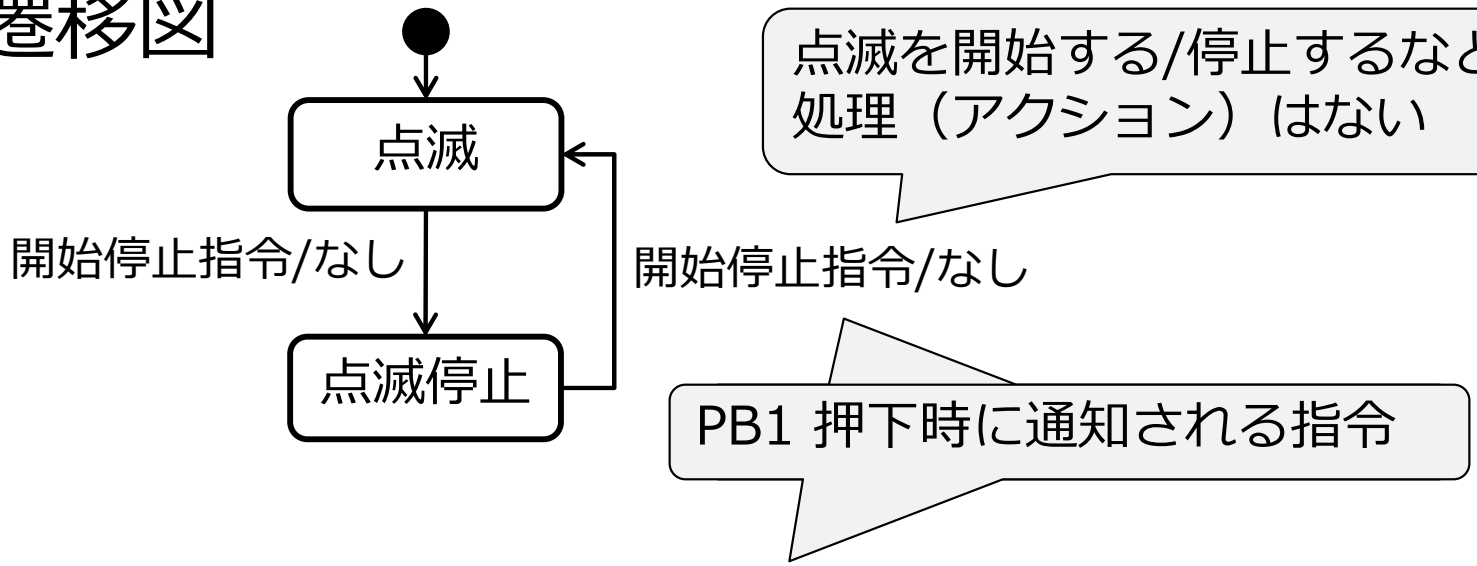
タスクの構造

- 点滅用タイマー（1000ミリ秒周期で起動）
  - 点滅タスクに「点滅」を通知
- PB1押下の割込み処理（PB1押下で起動）
  - 点滅タスクに「開始停止」を通知
- 点滅タスク
  - 受取った通知と状態によって動作



状態遷移

- 点滅の状態 ＝ [点滅 | 点滅停止]
- 状態遷移図



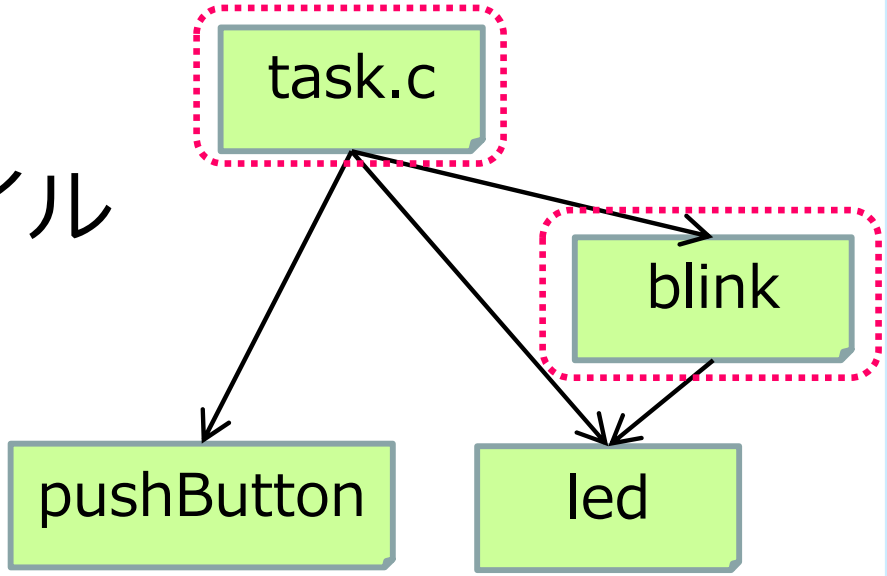
- 状態遷移表

点滅の状態 ＼ トリガ	開始停止指令
点滅	点滅停止 なし
点滅停止	点滅 なし

ファイルの構造

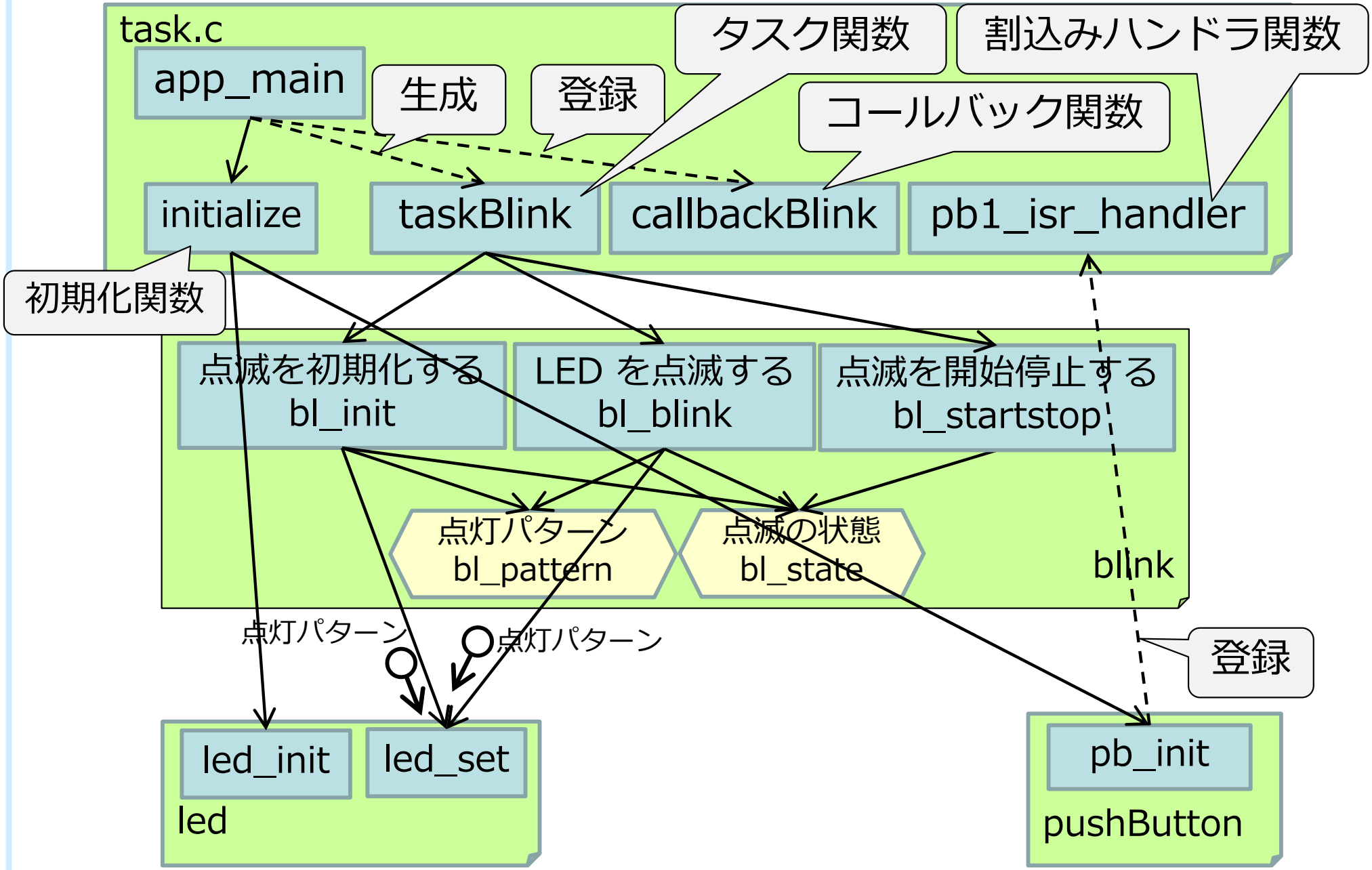
- LED 点滅に関連するファイル

  例題5 の LED 点滅に
  - 状態を保存する変数追加
  - 状態を変更する関数追加
  - 各関数に状態に関する処理追加



ファイル	責務
task.c	システム動作 ・タスクやタイマーの生成（app_main 関数） ・タスク関数、タイマーのコールバック関数、割込みハンドラ関数 ・初期化関数
blink	LED 点滅
led	LED 出力
pushButton	プッシュボタン入力

ファイルと関数の構造

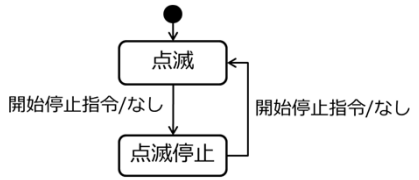

task.c
app_main
生成
登録
タスク関数
割込みハンドラ関数
コールバック関数
initialize
taskBlink
callbackBlink
pb1_isr_handler
初期化関数
点滅を初期化する
bl_init
LED を点滅する
bl_blink
点滅を開始停止する
bl_startstop
点灯パターン
bl_pattern
点滅の状態
bl_state
blink
点灯パターン
点灯パターン
登録
led_init
led_set
led
pb_init
pushButton

使用する API

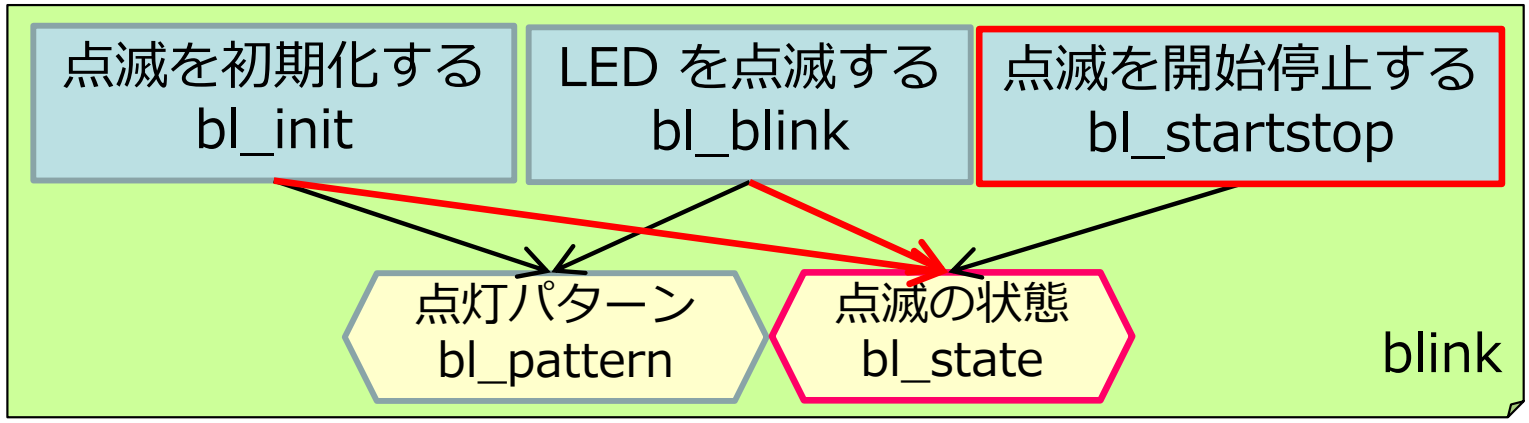
- タスクの生成（既出）
- ソフトウェアタイマーの生成と開始（既出）
- タスクへの通知（既出）
  - 通知値を使用する

API（Application Programming Interface）とは
- 異なるソフトウェア間で機能を共有するための仕組みのこと
- この授業では主に FreeRTOS の機能を利用するための関数のことを表す
  （例題では他にも gpio_set_level などマイコン ESP32 利用のための API も使用している）

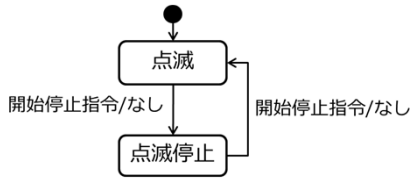
ファイル blink の変更



- 変数と関数を追加
  - 点滅の状態を表す変数 bl_state
  - 点滅を開始停止する関数 bl_startstop
    - 点滅の状態を変更する
- 点滅の状態に対応するように関数を変更
  - 点滅を初期化する → 最初の状態を設定
  - LED を点滅する → 点滅の状態が「点滅」のとき点滅



状態を表す変数



- 変数の値は 「点滅停止」 または 「点滅」
  - 点滅の状態 = [点滅停止 | 点滅]
- 列挙型 bl_state_t を使う

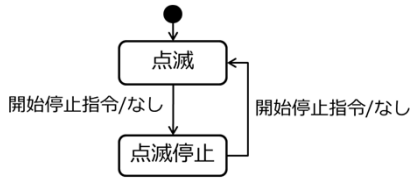
```
typedef enum {
    eBL_STOP,
    eBL_RUN
} bl_state_t;
```

  - 型 bl_state_t で宣言された変数は、eBL_STOP または eBL_RUN のどちらかの値を代入できる
- 状態を表す変数 bl_state の宣言

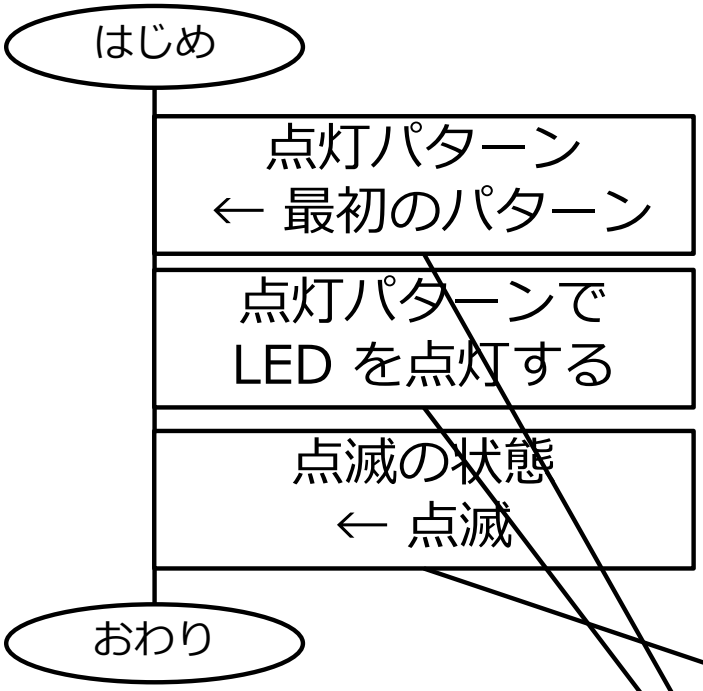
```
static bl_state_t       bl_state;
```

bl_state_t 型の変数 bl_state の値は
eBL_STOP （点滅停止）
eBL_RUN （点滅）のいずれか

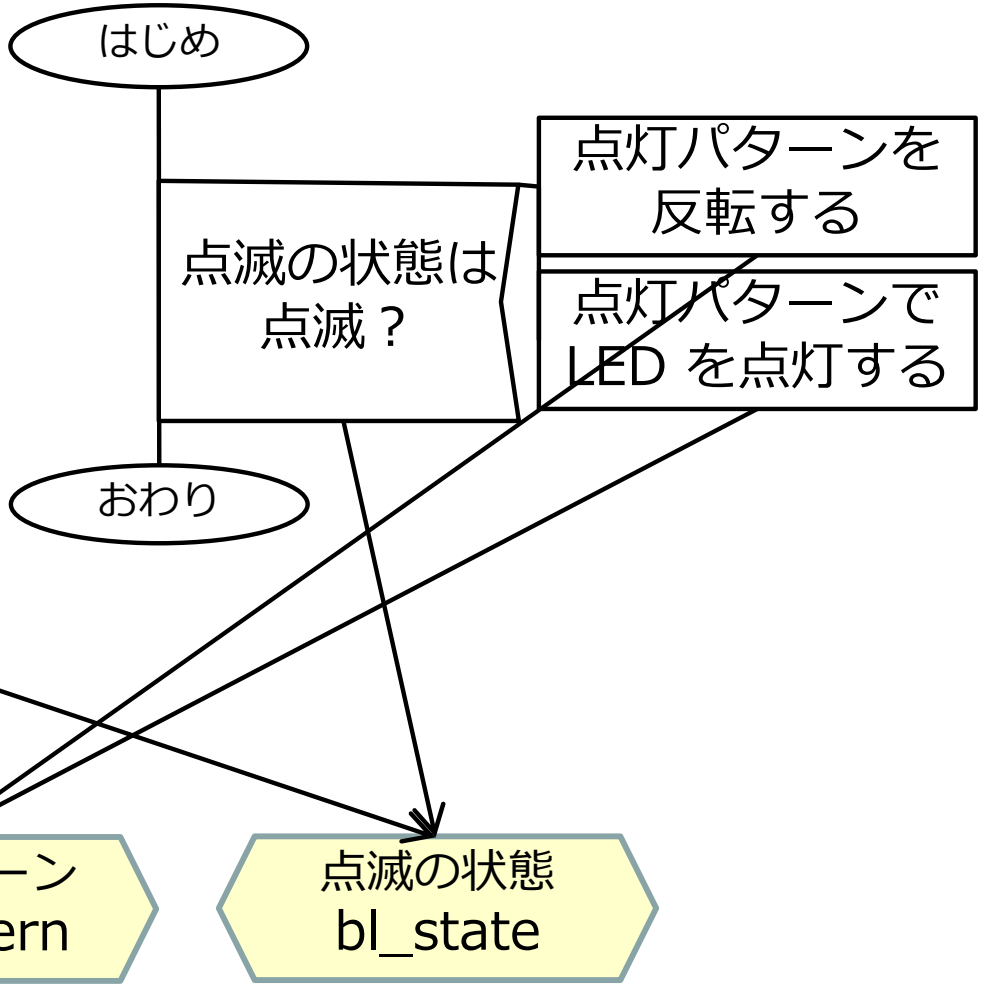
点滅の初期化と点滅



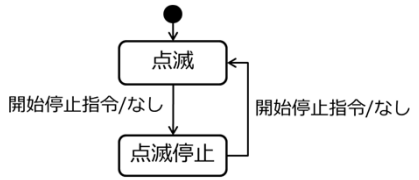
■ 点滅を初期化する



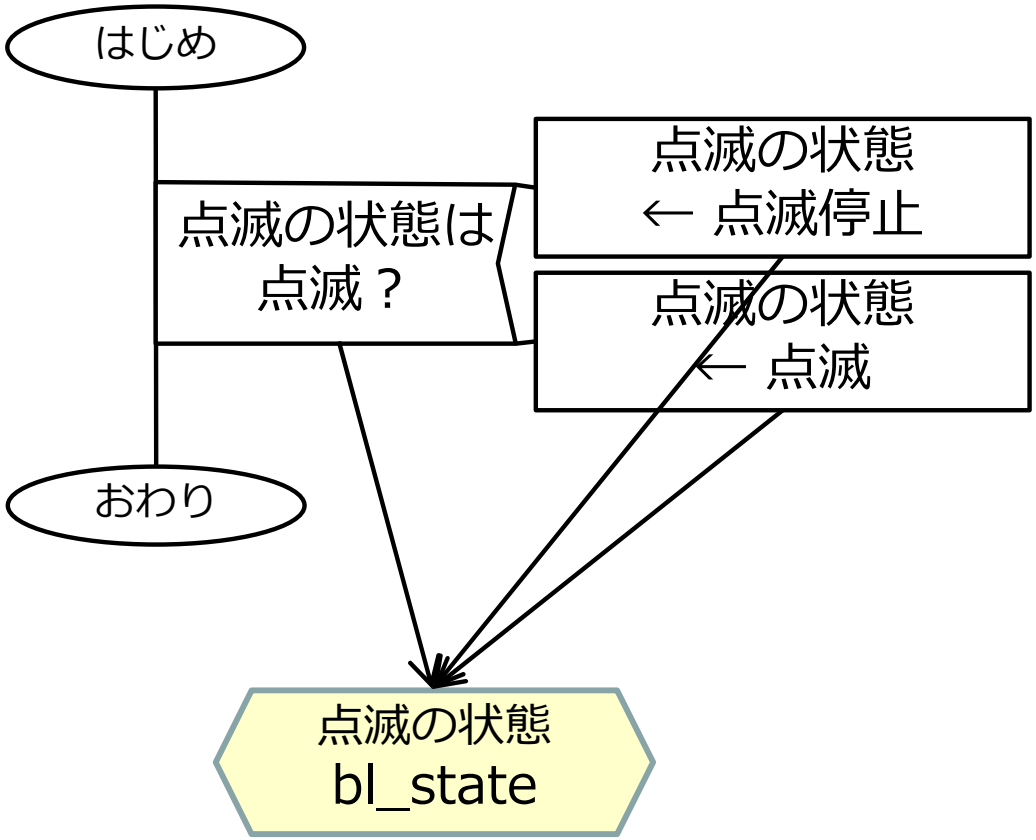
■ 点滅する



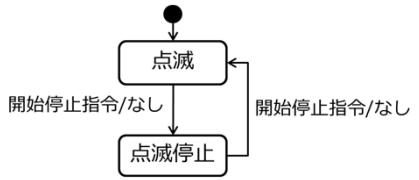
点滅を開始停止する



- 状態の遷移と遷移時のアクションを実行する
  - この例題では遷移時のアクションなし



点滅のプログラム



- 関数 bl_init

```
void bl_init(void)
{
    bl_pattern = BL_INIT;
    led_set(bl_pattern);
    bl_state = eBL_RUN;
    return;
}
```

- 関数 bl_startstop

```
void bl_startstop(void)
{
    if (bl_state == eBL_RUN) {
        bl_state = eBL_STOP;
    } else {
        bl_state = eBL_RUN;
    }
    return;
}
```

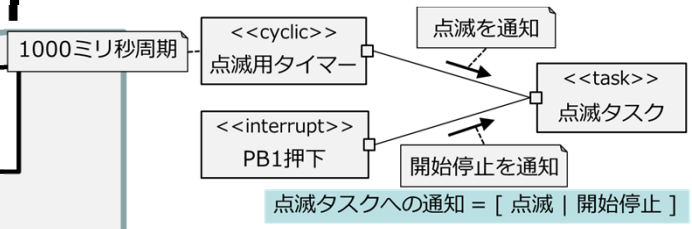
- 関数 bl_blink

```
void bl_blink(void)
{
    if (bl_state == eBL_RUN) {
        bl_pattern = bl_pattern ^ LOWER_4BIT;
        led_set(bl_pattern);
    }
    return;
}
```

ファイル task.c のポイント

```
// --- Header files (system)
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/timers.h"
// --- Header files (project)
#include "led.h"
#include "sevenSegmentLed.h"
#include "pushButton.h"
#include "blink.h"

// --- macros
// task
#define STACK_DEPTH     ((uint32_t) 409
#define PRIORITY_BLINK  (tskIDLE_PRIORIT
// notify
#define CLEAR_NONE      ((uint32_t) 0)
#define CLEAR_ALL       ULONG_MAX
#define PERIOD_BLINK    pdMS_TO_TICKS(1000)
#define TICKS_TO_WAIT   pdMS_TO_TICKS(1000 * 6
#define VALUE_NONE      ((uint32_t) 0)
#define VALUE_BLINK     ((uint32_t) 0x01)
#define VALUE_STARTSTOP ((uint32_t) 0x02)
```

#include 、マクロ



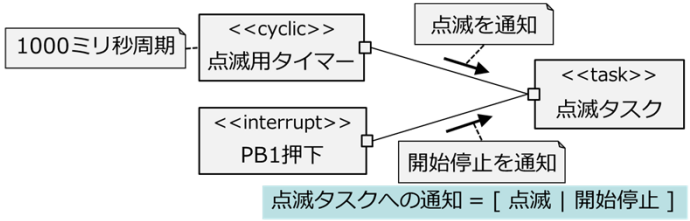
ソフトウェアタイマーが
コールバック関数を起動する周期

通知値に設定するビット

変数、プロトタイプ

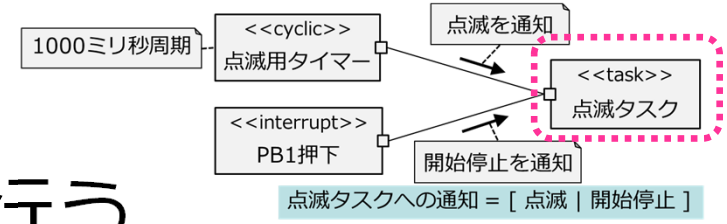
```
// --- data (static)
static TaskHandle_t     taskHandleBlink = NULL;
static TimerHandle_t    timerHandleBlink = NULL;

// --- prototypes (static)
static void taskBlink(void *arg);
static void callbackBlink(TimerHandle_t timer);
static void initialize(void);
```

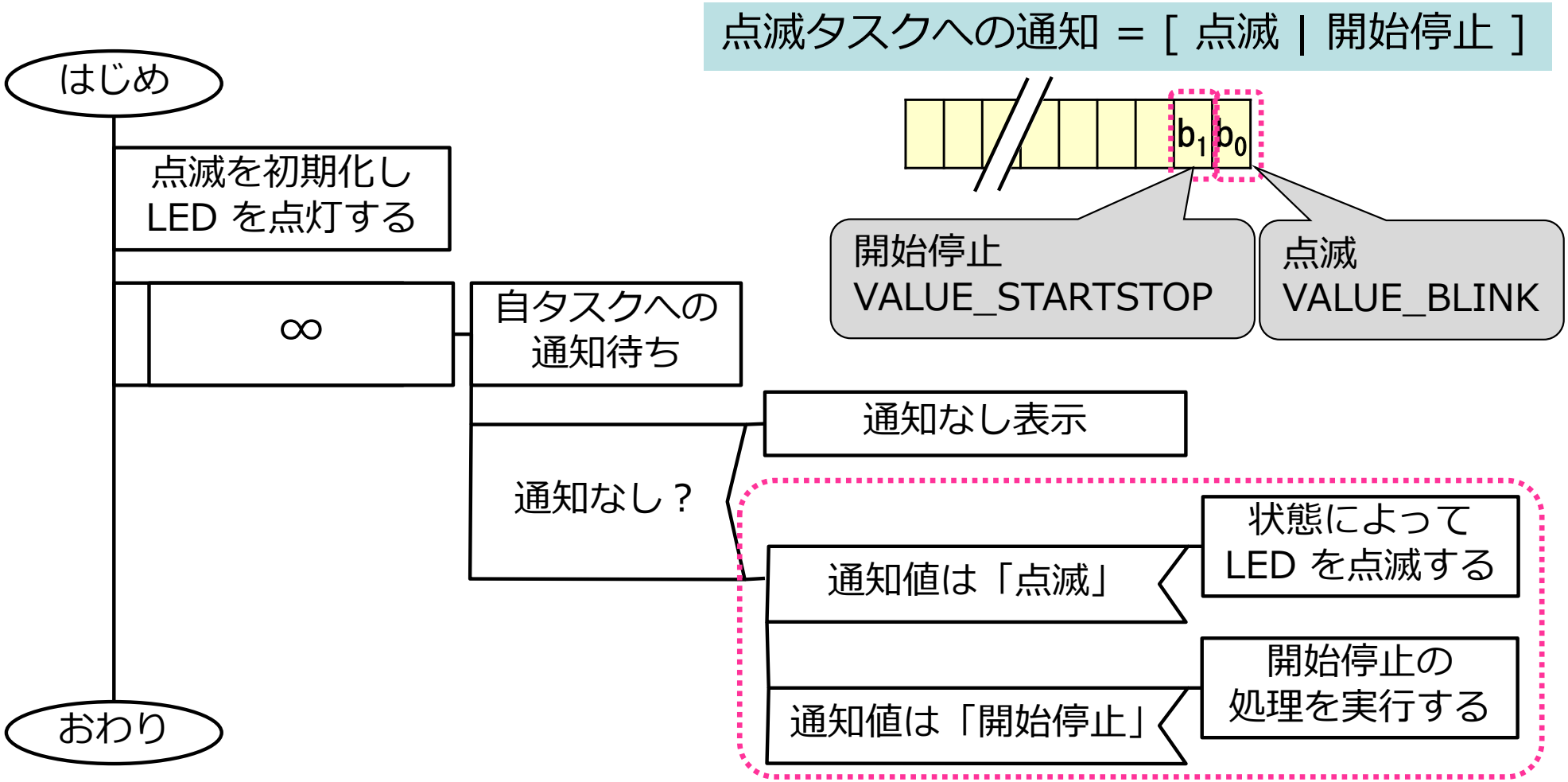
app_main 関数のアルゴリズム



点滅タスクのアルゴリズム

- 通知値によって
  点滅または開始停止の処理を行う





タスク関数のプログラム

```
static void taskBlink(void *arg)
{
    // monitor (省略)
    bl_init();
    for (;;) {  // closed loop
        BaseType_t  pd;
        uint32_t    notifiedValue;

        pd = xTaskNotifyWait(
            CLEAR_NONE,
            CLEAR_ALL,
            &notifiedValue,
            TICKS_TO_WAIT
        );
        if (pd != pdPASS) {
            puts("not notified");
        } else {
            if ((notifiedValue & VALUE_BLINK) != VALUE_NONE) {
                bl_blink();
            }
            if ((notifiedValue & VALUE_STARTSTOP) != VALUE_NONE) {
                bl_startstop();
            }
        }
    }
}
```

通知値を代入するための変数

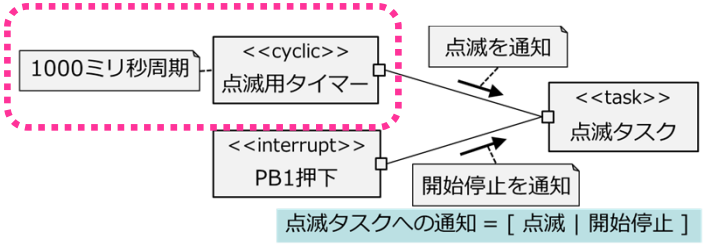
ブロック状態が解除されたとき
すべてのビットをクリアする

通知値を受け取る

通知値のビット0
（VALUE_BLINK）を調べる

通知値のビット1
（VALUE_STARTSTOP）を調べる

ソフトウェアタイマーのコールバック関数

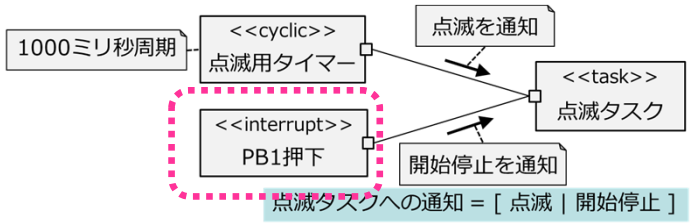


- アルゴリズム

- プログラム

```
static void callbackBlink(TimerHandle_t timer)
{
    (void)xTaskNotify(
        taskHandleBlink,
        VALUE_BLINK,
        eSetBits
    );
    return;
}
```

割込みハンドラ関数



- アルゴリズム

- プログラム

```
void pb1_isr_handler(void *arg)
{
    BaseType_t  higherPriorityTaskWoken = pdFALSE;

    (void)xTaskNotifyFromISR(
        taskHandleBlink,
        VALUE_STARTSTOP,
        eSetBits,
        &higherPriorityTaskWoken
    );
    portYIELD_FROM_ISR(higherPriorityTaskWoken);

    return;
}
```