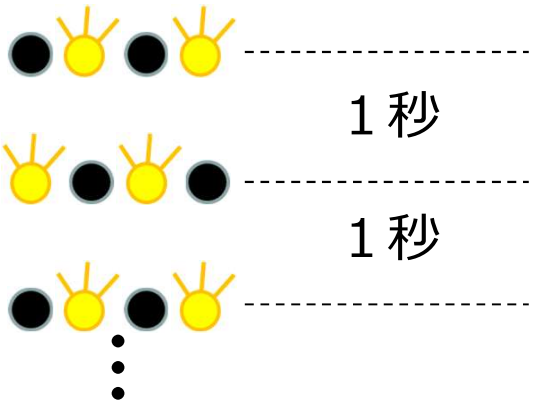
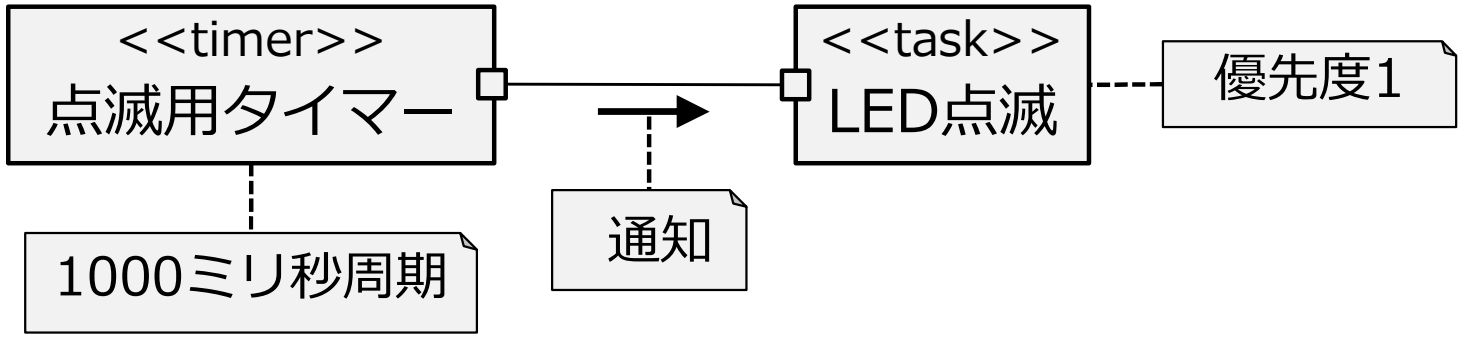
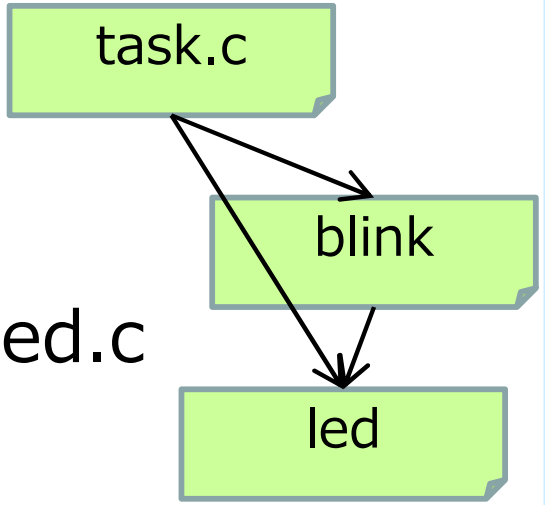
例題5　ソフトウェアタイマー

- 1 秒周期で LED の点灯パターンを反転する





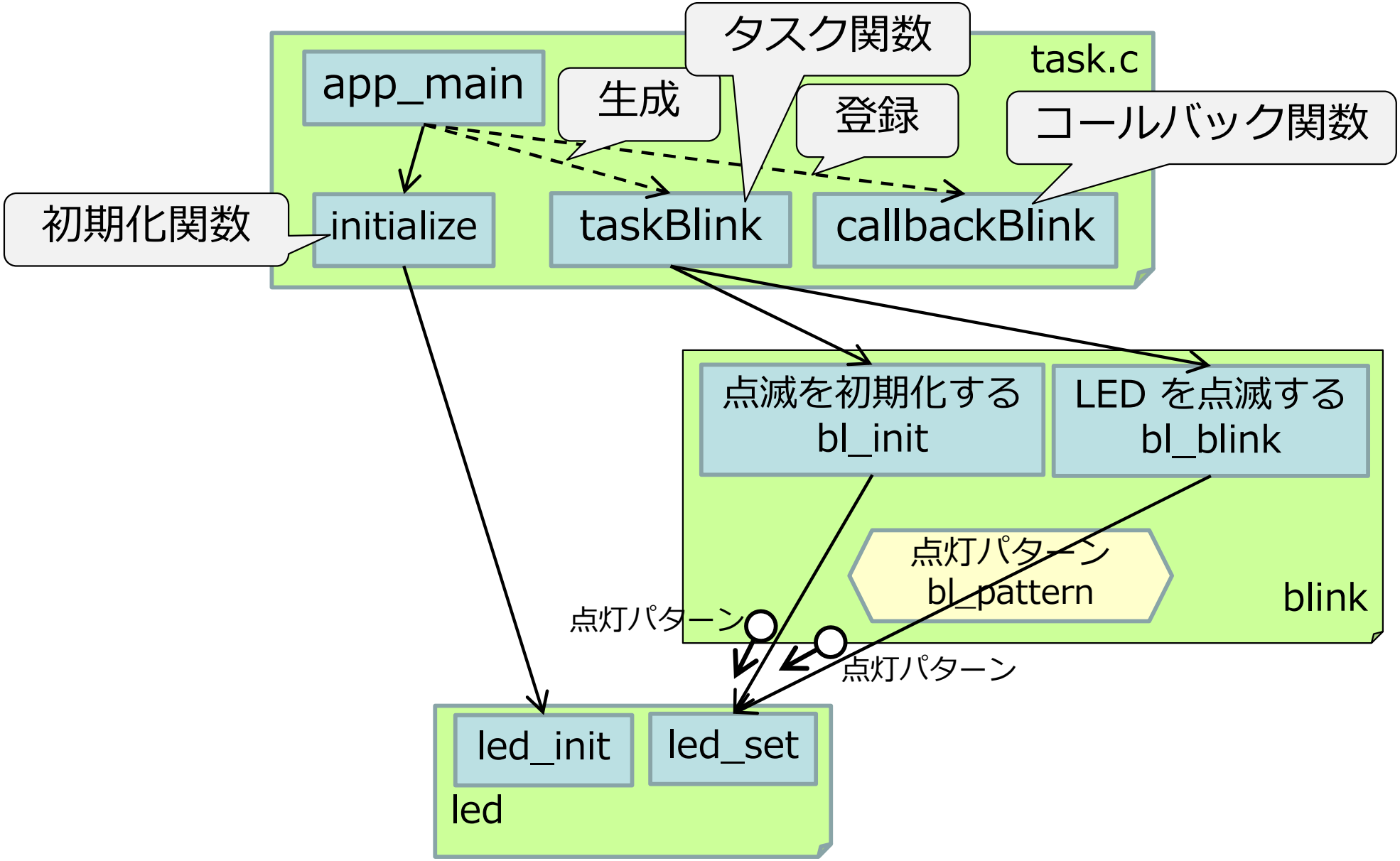
ファイルの構造



- 例題3 と同じファイル
  - LED 点滅：blink.h , blink.c , led.h , led.c
- 例題3 と異なるファイル
  - システム動作：task.c

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 ・初期化関数
blink	LED 点滅
led	LED 出力

ファイルと関数の構造


task.c
app_main
生成
タスク関数
登録
コールバック関数
初期化関数
initialize
taskBlink
callbackBlink
点滅を初期化する
bl_init
LED を点滅する
bl_blink
点灯パターン
bl_pattern
blink
点灯パターン
点灯パターン
led_init
led_set
led

使用する API

- タスクの生成（省略）
- ソフトウェアタイマー
  - xTimerCreate
    ソフトウェアタイマーを生成しコールバック関数を登録する
  - xTimerStart
    ソフトウェアタイマーを開始する
- タスクへの通知
  - xTaskNotifyWait
    自タスクへの通知を待つ（ブロック状態に遷移）
  - xTaskNotify
    タスクに通知する

ソフトウェアタイマーの生成

xTimerCreate

引数で指定される情報に基づいてソフトウェアタイマーを生成する

- 形式

```
TimerHandle_t xTimerCreate
(const char *const          pcTimerName,
 const TickType_t           xTimerPeriodInTicks,
 const UBaseType_t          uxAutoReload,
 void *const                pvTimerID,
 TimerCallbackFunction_t    pxCallbackFunction)
```

- 返却値
  - 生成されたソフトウェアタイマーのハンドル
  - NULL　　生成に失敗したとき

パラメータ

xTimerCreate

パラメータ		指定する内容
pcTimerName	名前	デバッグなどで使用できる名前（RTOS は使用しない）
xTimerPeriodInTicks	タイマーの周期	ティック数で指定
uxAutoReload	繰返しの指定	pdTRUE　繰返し起動 pdFALSE　一度だけ起動
pvTimerID	タイマーの識別子	コールバック関数などで参照するソフトウェアタイマーの識別子
pxCallbackFunction	コールバック関数	周期に到達したときに呼び出される関数

ソフトウェアタイマーの開始

xTimerStart

生成されているタイマーを開始する

- 形式

```
BaseType_t xTimerStart
(TimerHandle_t     xTimer,
 TickType_t        xTicksToWait)
```

- 返却値
  - pdPASS　成功したとき
  - pdFAIL　タイマーを開始できなかったとき
- パラメータ
  - xTimer　開始するタイマーのハンドル
  - xTicksToWait　タイマーを開始したタスクが
ブロック状態でタイマーの開始を待つ
最大待ち時間（ティック数で指定）

自タスクへの通知待ち

xTaskNotifyWait

自タスクへの通知を待つ

- 形式

```
BaseType_t xTaskNotifyWait
(uint32_t   ulBitsToClearOnEntry,
 uint32_t   ulBitsToClearOnExit,
 uint32_t   *pulNotificationValue,
 TickType_t xTicksToWait)
```

- 返却値
  - pdPASS　通知を受理した
  - pdFAIL　通知を受理していない

パラメータ

xTaskNotifyWait

パラメータ		指定する内容
ulBitsToClearOnEntry	待ち開始時にクリアするビット	通知値のビットのうち待ち開始時にクリアするビット（補足参照）
ulBitsToClearOnExit	待ち終了時にクリアするビット	通知値のビットのうち待ち終了時にクリアするビット（補足参照）
pulNotificationValue	待ち解除時の値を代入する領域	待ちが解除されたときの通知値を代入する領域（呼出しもとに返す）
xTicksToWait	最大待ち時間	ブロック状態で通知を待つ最大待ち時間ティック数で指定

補足）　タスクは、内部に通知を受け取るためのデータ（通知値/notifications）を持っている
xTaskNotifyWait は待ち開始/終了時に、通知値をビット単位でクリアできる（例題では使用していない）

タスクへの通知

xTaskNotify

タスクに通知する

- 形式

```
BaseType_t xTaskNotify
(TaskHandle_t   xTaskToNotify,
 uint32_t       ulValue,
 eNotifyAction  eAction )
```

- 返却値
  - pdPASS　通知は通知先のタスクに受理された
    引数 eAction が eNoAction/eSetBit/eIncrement
    のとき、返却値は常に pdPASS

パラメータ

パラメータ		指定する内容
xTaskToNotify	タスクのハンドル	通知先のタスクのハンドル
ulValue	通知に関する値	RTOS によって値がどのように使われるかは、引数 eAction との関連で決まる （eNoAction や eIncrement では この引数は使われない）
eAction	通知時のアクション	eNoAction 何もしない eSetBits ulVasue のビットを設定する eIncrement 通知値を 1 増やす など

ファイル task.c のポイント

- ヘッダファイルの #include

```
// --- 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 "blink.h"
```

ソフトウェアタイマーを使用するため

- マクロの定義

```
// --- macros
// task
#define STACK_DEPTH      ((uint32_t) 4096)
#define PRIORITY_BLINK   (tskIDLE_PRIORITY + 1)
#define PERIOD_BLINK     pdMS_TO_TICKS(1000)
// notify
#define CLEAR_NONE       ((uint32_t) 0)
#define VALUE_NONE       ((uint32_t) 0)
#define TICKS_TO_WAIT    pdMS_TO_TICKS(1000 * 10)
```

ソフトウェアタイマーを起動する周期

通知値に関するマクロ

通知を待つ時間

ハンドルの変数とプロトタイプ

```
// --- 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);
```

タスクのハンドルを代入する変数

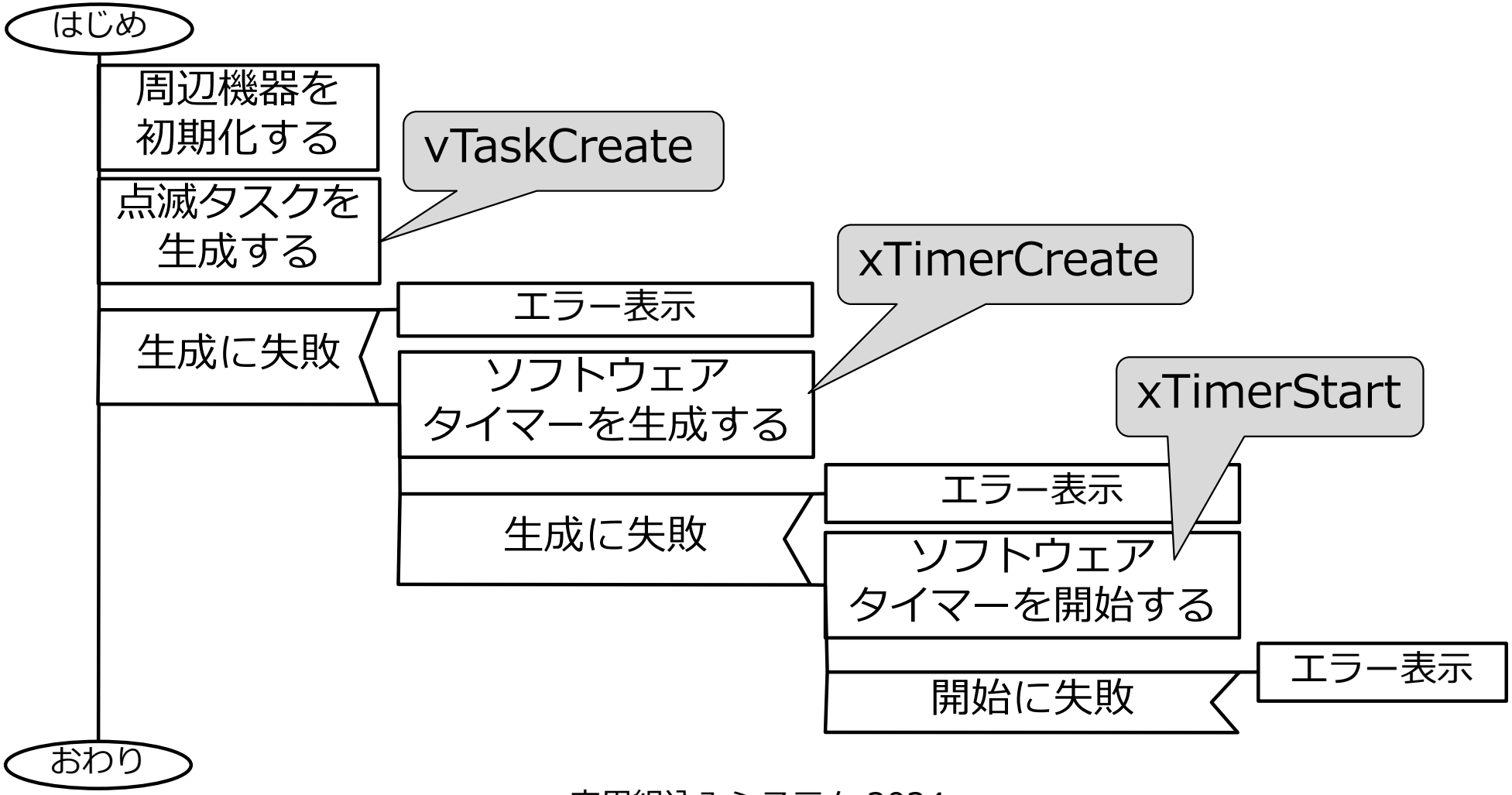
タイマーのハンドルを代入する変数

タスク関数 のプロトタイプ

ソフトウェアタイマーの
コールバック関数のプロトタイプ

app_main 関数のアルゴリズム

- 周辺機器を初期化しタスクを生成して終了する
  - 終わらない繰返しを実行しない



app_main 関数

```
void app_main(void)
{
    BaseType_t  pass;

    // initialize devices
    initialize();
    // create task
    pass = xTaskCreate(
        &taskBlink,
        "taskBlink",
        STACK_DEPTH,
        NULL,
        PRIORITY_BLINK,
        &taskHandleBlink
    );
    if (pass != pdPASS) {
        // エラー表示（省略）
    } else {
        // create timer
        timerHandleBlink = xTimerCreate(
            "timerBlink",
            PERIOD_BLINK,
            pdTRUE,
            NULL,
            &callbackBlink
        );
        if (timerHandleBlink == NULL) {
            // エラー表示（省略）
        } else {
            pass = xTimerStart(
                timerHandleBlink, 0);
            if (pass != pdPASS) {
                // エラー表示（省
            }
        }
    }
    return;
}
```

点滅タスクの生成

タスク関数 taskBlink のアドレス

タスクのハンドルを代入する領域

タイマーのハンドル

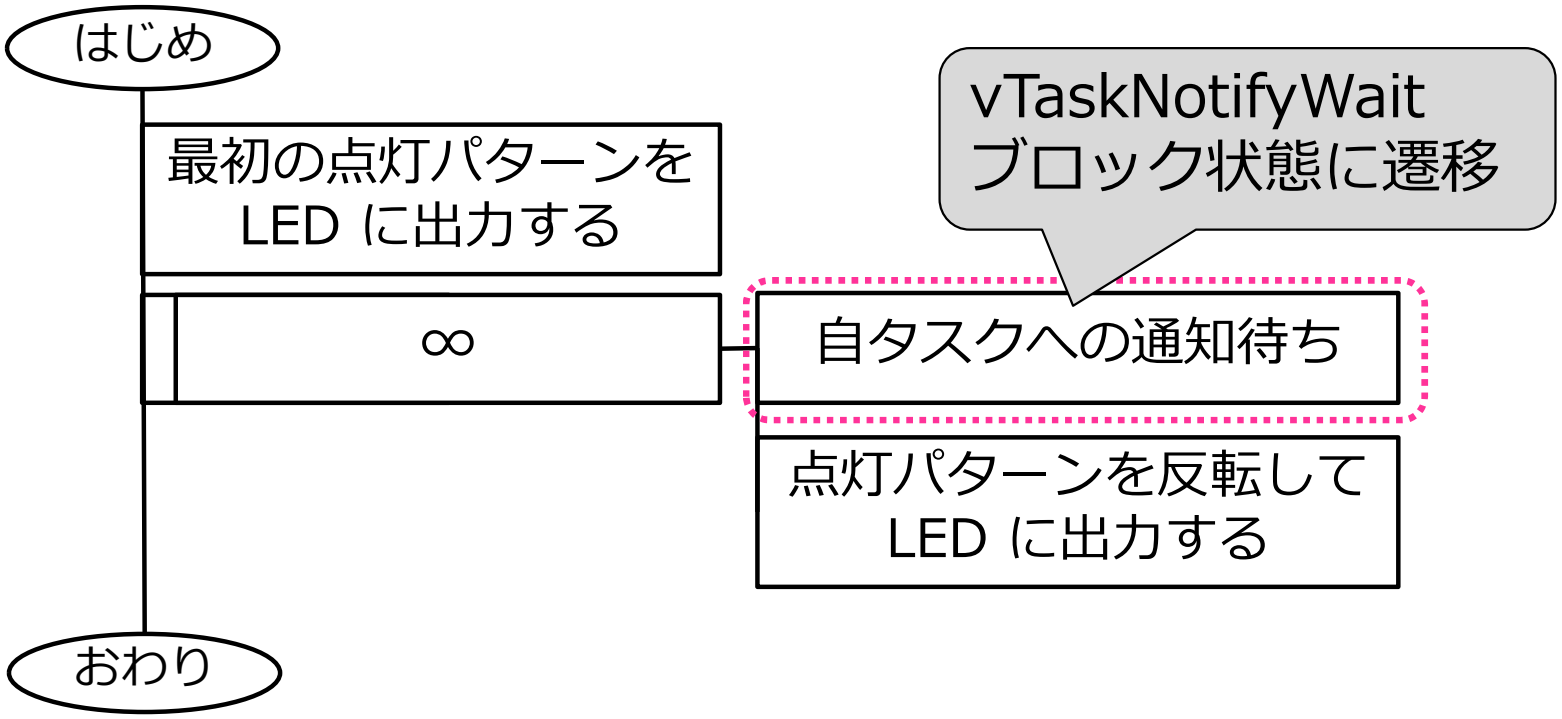
ソフトウェアタイマーの生成

コールバック関数 callbackBlink のアドレス

ソフトウェアタイマーの開始（タイマーのハンドルを指定）

タスク関数のアルゴリズム

- 自タスクへの通知を待ち（ブロック状態に遷移）、通知があれば点灯パターンを反転するこれを繰り返す



タスク関数 taskBlink

```
static void taskBlink(void *arg)
{
    // monitor 動作確認のための記述（省略）

    bl_init();
    for (;;) {  // closed loop
        BaseType_t  pd;

        pd = xTaskNotifyWait(
            CLEAR_NONE,
            CLEAR_NONE,
            NULL,
            TICKS_TO_WAIT
        );
        if (pd != pdPASS) {
            // エラー表示
        } else {
            bl_blink();
        }
    }
}
```

最初の点灯パターンを
LED に出力する

自タスクへの通知を待つ
（ブロック状態に遷移）

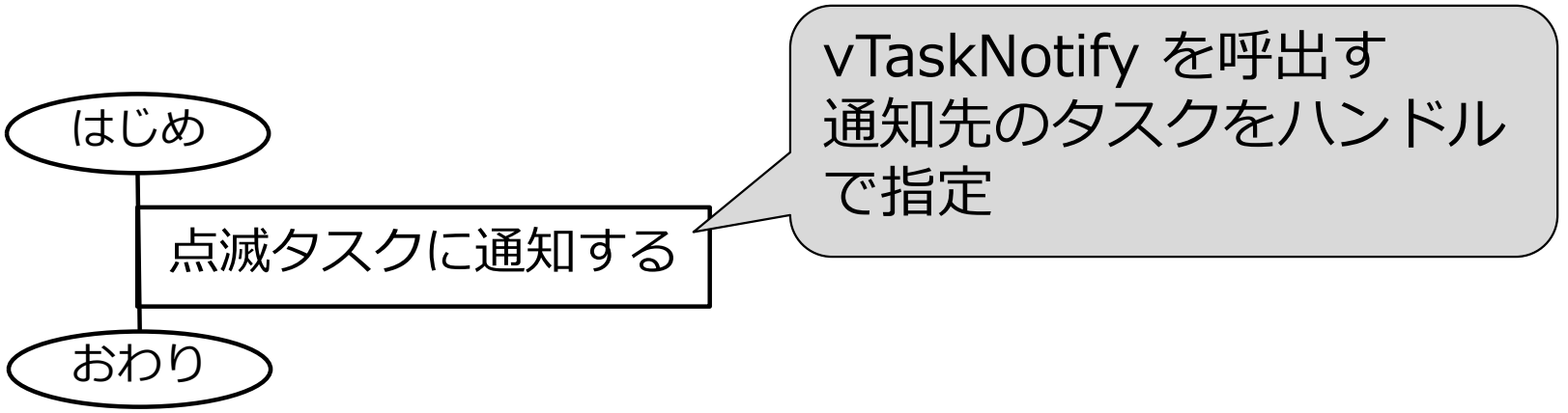
通知値を使うときに使用
（この例題では通知値を使わない）

通知を待つ最大時間

点灯パターンを反転してLED に出力する

コールバック関数のアルゴリズム

- 点滅タスクに通知する



ソフトウェアタイマーのコールバック関数では
必要最小限の処理を実行して終了する

コールバック関数 callbackBlink

```
static void callbackBlink(TimerHandle_t timer)
{
    (void)xTaskNotify(taskHandleBlink, VALUE_NONE, eNoAction);
    return;
}
```

点滅タスクに通知する

通知先のタスク（点滅タスク）のハンドル

通知に関する値（この例題ではなし）

通知時のアクション（この例題では何もしない）

参考)
関数呼出し式のキャスト（void）は
「返却値を使わない」ことを表す
API xTaskNotify の返却値は
第三引数が eNoAction のとき常に pdPASS