

例題4 二つのタスク

■ 二つのプログラムを並行動作させる

➤ LED 点滅 (例題3)

◆ 1000 ミリ秒間隔で LED を点滅する

➤ インクリメント

◆ 250 ミリ秒間隔で数を数え、十六進数表記で
右の 7 セグメント LED に表示する
(左の 7 セグメント LED はゼロ)



優先度低 (1)

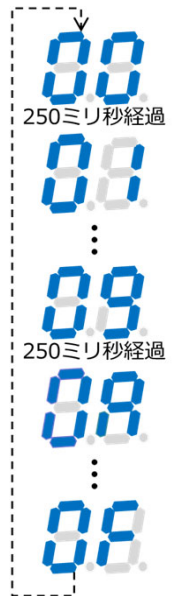
<<task>>
LED点滅

1000ミリ秒 (1 秒) 遅延

優先度高 (2)

<<task>>
インクリメント

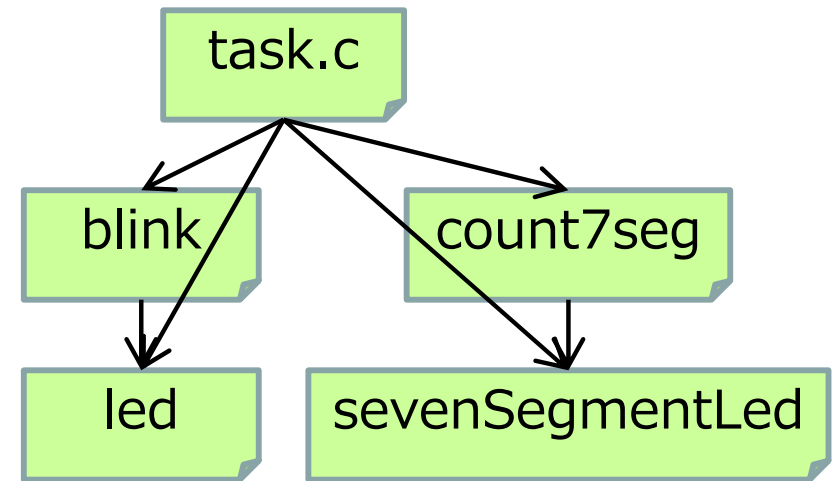
250ミリ秒 (0.25秒) 遅延



ファイルの構造

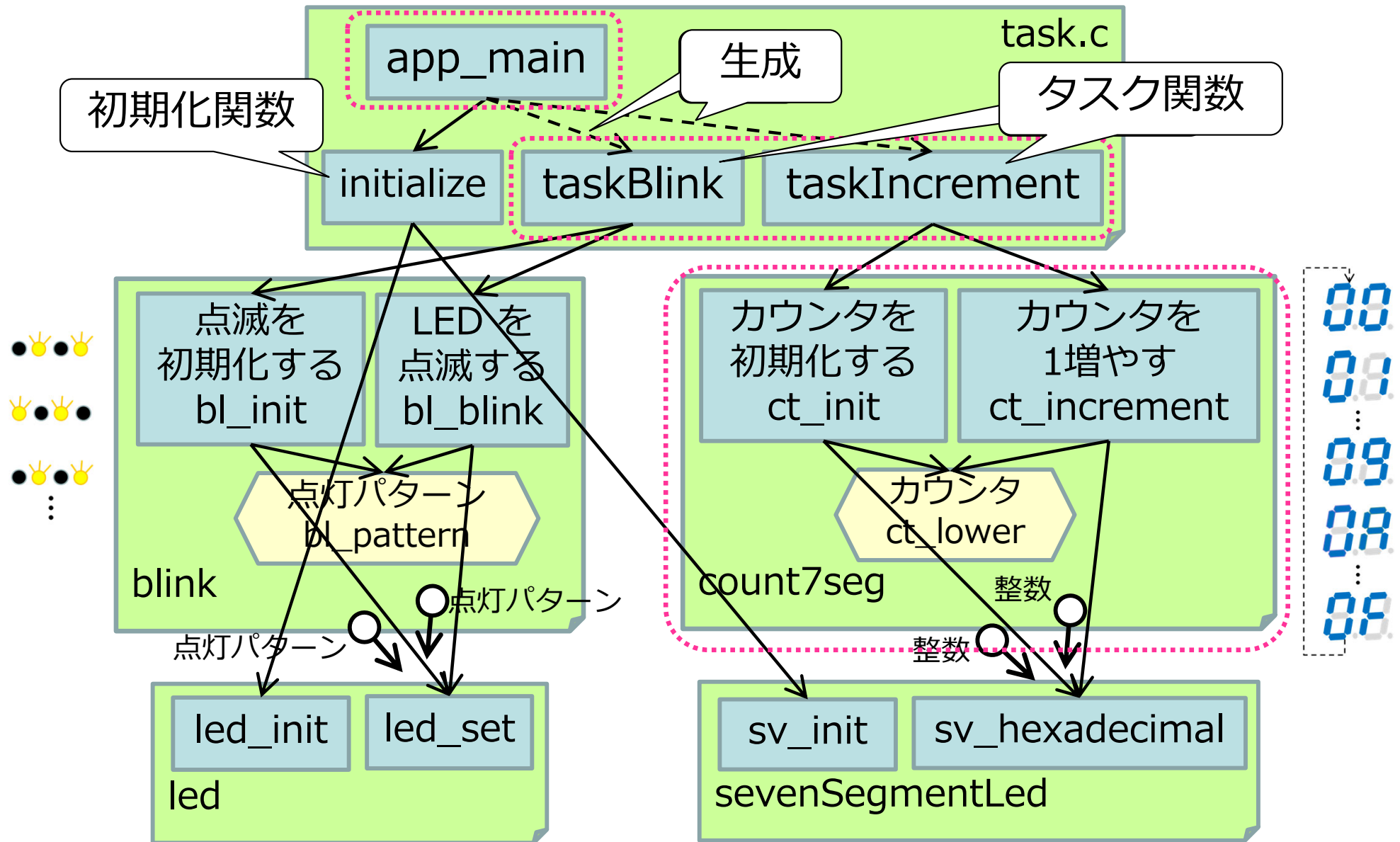
- 例題3 のファイルに
カウントに関連する
ファイルを追加

- count7seg.h と count7seg.c
- sevenSegmentLed.h と sevenSegmentLed.c



ファイル	責務
task.c	システム動作 (タスクの生成、タスク関数、初期化)
count7seg	計数 (7セグメントLED表示)
sevenSegmentLed	7セグメントLED表示
blink	LED 点滅
led	LED 出力

ファイルと関数の構造



使用する API

■ タスクの生成

➤ xTaskCreate

タスク関数などを指定してタスクを生成する

■ 自タスクの遅延

➤ vTaskDelay

指定された時間、タスクをブロック状態に遷移し
遅延させる

例題3と同じ

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"
// --- Header files (project)
#include "led.h"
#include "sevenSegmentLed.h"
#include "blink.h"
#include "count7seg.h"
```

点滅のため

カウントのため

■ マクロの定義

```
// --- macros
#define STACK_DEPTH ((uint32_t) 4096)
#define PRIORITY_BLINK (tskIDLE_PRIORITY + 1) // lower
#define PRIORITY_INCREMENT (tskIDLE_PRIORITY + 2) // higher
#define DELAY_BLINK pdMS_TO_TICKS(1000)
#define DELAY_INCREMENT pdMS_TO_TICKS(250)
```

点滅タスクの優先度（低）
アイドルタスクは優先度ゼロ

カウントタスクの優先度（高）

lower
higher

それぞれのタスクの周期

タスクのハンドルとプロトタイプ

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

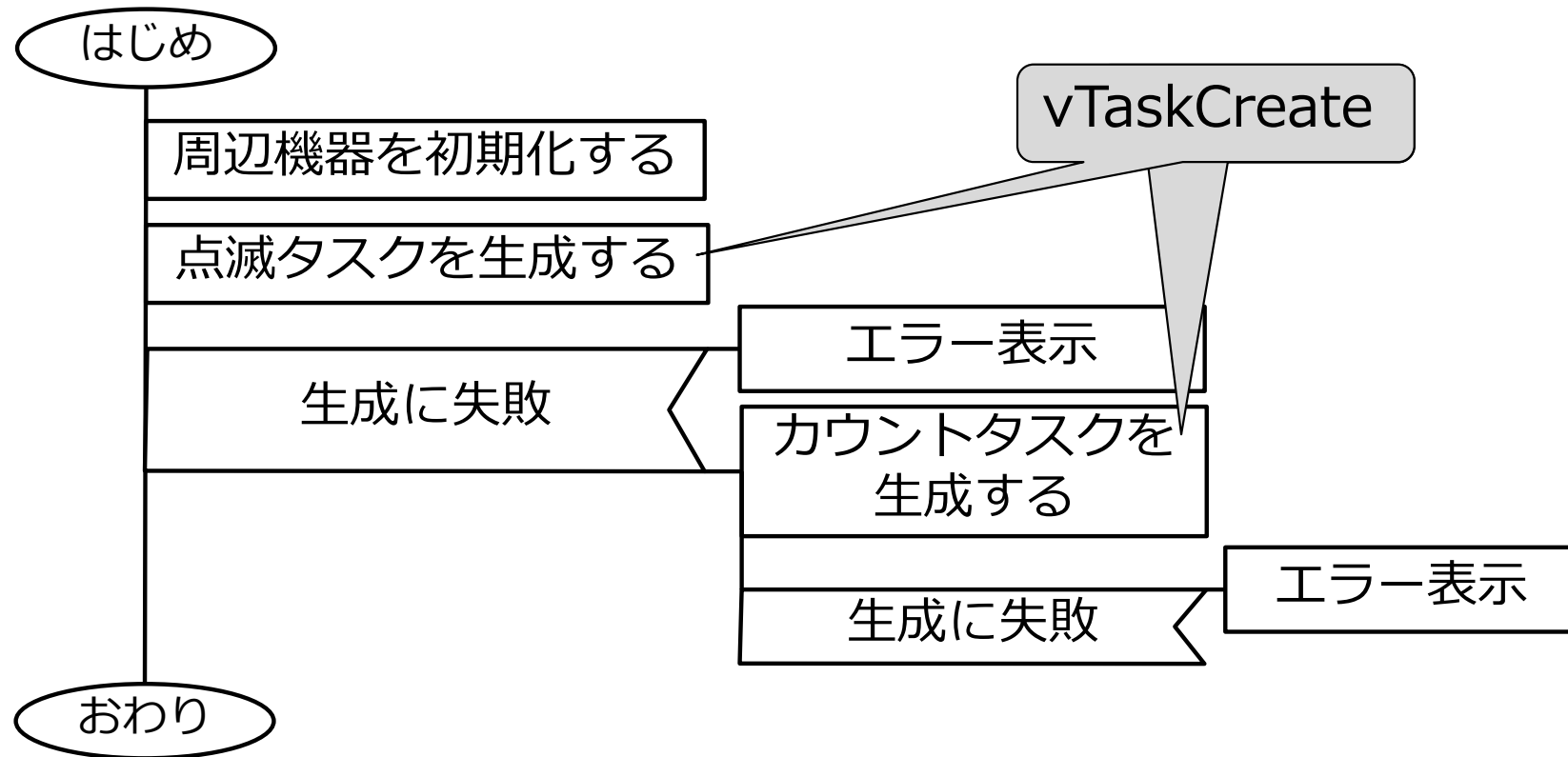
各タスクのハンドルを代入
する変数

```
// --- prototypes (static)
static void taskBlink(void *arg);
static void taskIncrement(void *arg);
static void initialize(void);
```

各タスクのタスク関数 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 device
    initialize();
    // create tasks
```

```
    pass = xTaskCreate(
        &taskBlink,
        "taskBlink",
        STACK_DEPTH,
        NULL,
        PRIORITY_BLINK,
        &taskHandleBlink
    );
```

```
    if (pass != pdPASS) {
        // エラー表示 (省略)
    } else {
```

点滅タスクの生成

タスク関数
taskBlink
のアドレス

関数 initialize

```
static void initialize(void)
{
    led_init();
    sv_init();
    return;
}
```

カウントタスクの生成

```
    pass = xTaskCreate(
        &taskIncrement,
        "taskIncrement",
        STACK_DEPTH,
        NULL,
        PRIORITY_INCREMENT,
        &taskHandleIncrement
    );
```

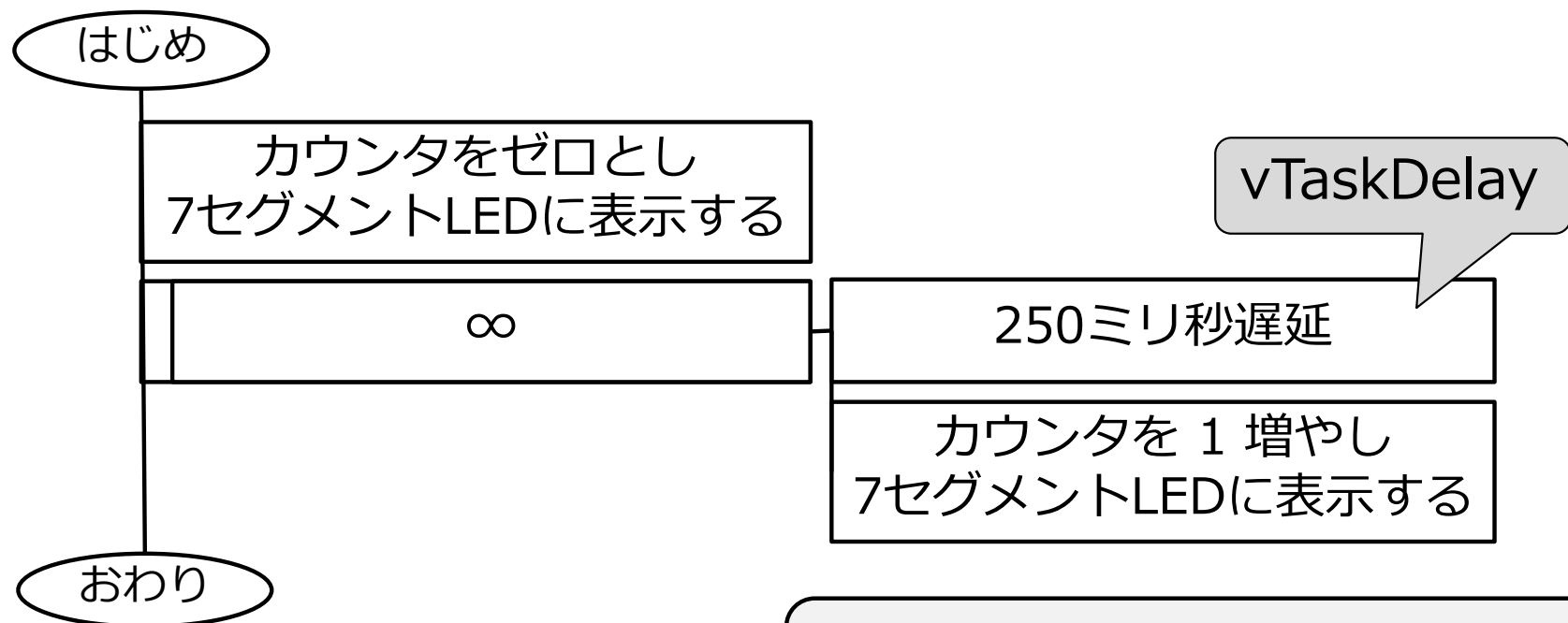
```
    if (pass != pdPASS) {
        // エラー表示 (省略)
    }
}
```

```
return;
```

タスク関数
taskIncrement
のアドレス

タスク関数のアルゴリズム taskIncrement

- 0.25 秒（250ミリ秒） 間隔でカウンタを 1 増やし 7セグメントLED に表示する
これを繰り返す



タスク関数は「終わらない繰り返し」
として作成（終了させない）

タスク関数 taskIncrement

```
static void taskIncrement(void *arg)
{
    // monitor
    TaskHandle_t handle = xTaskGetCurrentTaskHandle();
    char* taskname = pcTaskGetName(handle);
    puts(taskname);

    ct_init();
    for (;;) { // closed loop
        vTaskDelay(DELAY_INCREMENT);
        ct_increment();
    }
}
```

動作確認のための記述（後述）

カウンタをゼロとし
7セグメントLEDに表示する

タスクを 0.25秒（250ミリ秒）間
ブロック状態に遷移させる

カウンタを 1 増やし
7セグメントLEDに表示する

ファイル count7seg

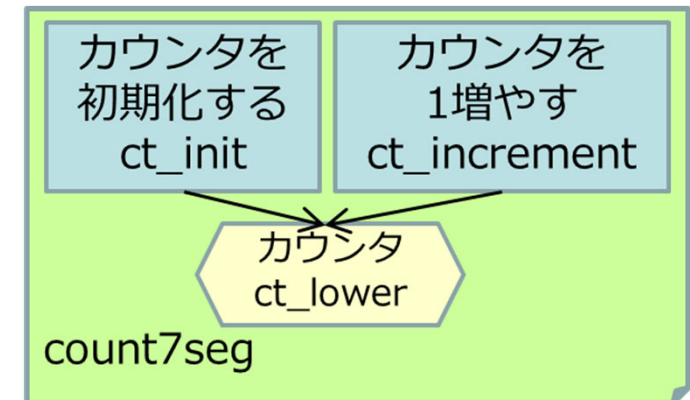
■ ヘッダファイル

```
extern void ct_init(void);  
extern void ct_increment(void);
```

■ 実装ファイル

➤ ヘッダファイルの #include

```
// --- Header files (system)  
#include <stdio.h>  
// --- Header files (project)  
#include "sevenSegmentLed.h"  
#include "led.h"  
// --- Header of own file  
#include "count7seg.h"
```



➤ マクロの定義

```
#define CT_OVER 0x10U
```

➤ データの定義

```
static unsigned char    ct_lower;
```

関数 ct_increment

■ 関数 ct_init

```
void ct_init(void)
{
    ct_lower = 0U;
    sv_hexadecimal(ct_lower);
    return;
}
```

■ 関数 ct_increment

```
void ct_increment(void)
{
    ct_lower = (ct_lower + 1U) % CT_OVER;
    sv_hexadecimal(ct_lower);
    return;
}
```

変数 ct_lower の値を 1 増やす
ただし CT_OVER (0x10) の剰余系
(0x0F を 1 増やすとゼロ)

カウンタ
ct_lower

参考：動作確認

- tools フォルダの monitor.bat を利用すると開発 PC にメッセージなどを出力できる

➤ 動作確認に利用できます

例) タスク関数 taskBlink

puts などを使うため

```
// --- Header files (system)
```

```
#include <stdio.h>
```

```
static void taskBlink(void *arg)
```

```
{
```

```
    // monitor
```

```
    TaskHandle_t handle = xTaskGetCurrentTaskHandle();
```

```
    char* taskname = pcTaskGetName(handle);
```

```
    puts(taskname);
```

```
    bl_init();
```

```
    for (;;) { // closed loop
```

```
        vTaskDelay(DELAY_BLINK);
```

```
        bl_blink();
```

```
    }
```

```
}
```

実行中のタスクのハンドルと (handle)
タスクの名前を取得する (taskname)

実行中のタスクの名前を表示する

参考：動作確認の操作

- ① ./build.bat
ビルド

```
PS C:\Espressif\frameworks\esp-idf-v5.0.2> cd C:\work\Sample04\tools
PS C:\work\Sample04\tools> ./build
```

- ② ./flash.bat com○
転送

```
or run 'idf.py -p (PORT) flash'
```

```
C:\work\Sample04>popd
PS C:\work\Sample04\tools> ./flash.bat com
```

- ③ ./monitor.bat com○
モニタリング（監視）

```
Hard resetting via RTS pin...
Done
```

```
C:\work\Sample04>popd
PS C:\work\Sample04\tools> ./monitor.bat com
```

puts の出力

```
I (265) heap_init: At 4008A990 len 00015670 (85 KiB): IRAM
I (271) heap_init: At 3FF80000 len 00002000 (8 KiB): RTCRAM
I (279) spi_flash: detected chip: generic
I (282) spi_flash: flash io: dio
W (286) spi_flash: Detected size(4096k) larger than the size in the bina
size in the binary image header.
I (300) cpu_start: Starting scheduler on PRO CPU.
taskIncrement
taskBlink
```

- ④ Control+]
モニタリング停止

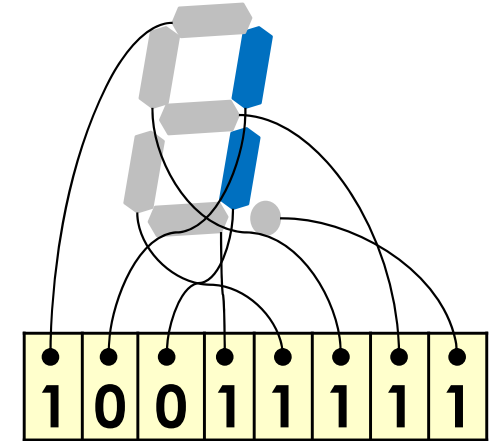
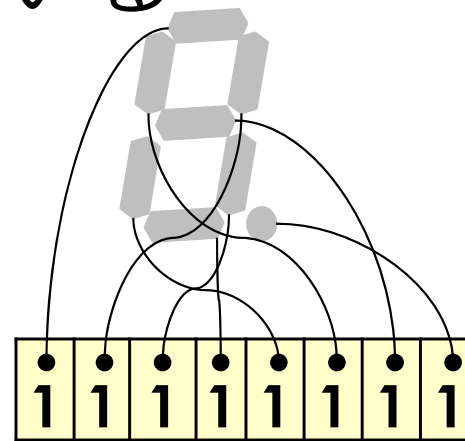
```
I (300) cpu_start: Starting scheduler on PRO CPU.
taskIncrement
taskBlink
```

```
C:\work\Sample04>popd
PS C:\work\Sample04\tools>
```

参考：7セグメントLED

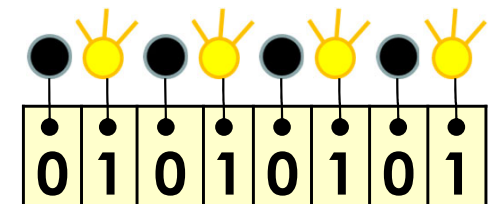
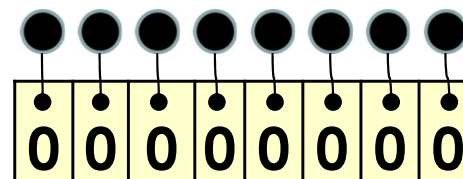
- 7セグメントLEDの各セグメントが、ポートの各ビットに対応している

- 0 点灯
- 1 消灯



- 8つのLEDの場合

- 1 消灯
- 0 点灯



参考：7セグメントLED の表示

■ ファイル sevenSegmentLed のヘッダファイル

```
#ifndef SEVEN_SEGMENT_LED_H
#define SEVEN_SEGMENT_LED_H
```

```
// --- prototypes (extern)
extern void sv_init(void);
extern void sv_hexadecimal(unsigned char number);
```

```
#endif // SEVEN_SEGMENT_LED_H
```

引数の8ビットのデータ (0~255) を
二つの7セグメントLEDに表示する

0 ~ F を構成する
8ビットのデータ

```
// =====
// File : sevenSegmentLed.c
// Role : seven segment LED
// Date : 2024.04.11
// Author : Osaka Sangyo University
// =====
// --- Header files (system)
#include "sdkconfig.h"
#include "hal/gpio_types.h"
#include "driver/gpio.h"
// --- Header of own file
#include "sevenSegmentLed.h"

// --- macros
#define SV_HEXADECIMAL 10
#define SV_DS GPIO_NUM_0
#define SV_SHCP GPIO_NUM_2
#define SV_STOP GPIO_NUM_4
#define SV_PATTERN_MAX 20
#define SV_SEGMENTS 8
#define SV_NUMBER 2

// --- static (static)
static const unsigned char sv_number_to_pattern[SV_PATTERN_MAX]
= {
    0x00, // 0 abcd ef
    0xF0, // 1 bc
    0x40, // 2 ab def
    0x80, // 3 abcd g
    0x90, // 4 bc fg
    0x20, // 5 a cde fg
    0x20, // 6 a cde fg
    0xF0, // 7 abc
    0x80, // 8 abcdefg
    0x80, // 9 abcdefg
    0x80, // A abcdefg
    0x80, // b cdefg
    0x00, // C a de g
    0xA0, // d bcde g
    0x80, // E a defg
    0x80, // F a efg
    0x00 // not digit
};

// --- prototypes (static)
static void sv_set(unsigned char pattern);
static void sv_fix(void);
```

関数 sv_hexadecimal の定義

```
// --- functions (extern)
// =====
// Name : sv_init
// Function : initialize seven segment LED
// Parameters : none
// Return : none
// notes : call before closed loop

// =====
// Name : sv_hexadecimal
// Function : display hexadecimal number on 7-segment LED
// Parameters : number
// Return : none
// notes : none

// =====
void sv_hexadecimal(unsigned char number)
{
    unsigned char num_1;
    unsigned char num_10;
    unsigned char lower;
    unsigned char upper;

    num_1 = number % SV_HEXADECIMAL;
    lower = sv_number_to_pattern[num_1];
    num_10 = (number / SV_HEXADECIMAL) % SV_HEXADECIMAL;
    upper = sv_number_to_pattern[num_10];
    sv_set(lower);
    sv_fix();
    sv_set(upper);
    sv_fix();
    return;
}
```

```
// --- functions (static)
// =====
// Name : sv_set
// Function : output pattern on seven segment LED
// Parameters : pat
// Return : none
// notes : none

// =====
static void sv_set(unsigned char pattern)
{
    unsigned char i;
    for (i = 0; i < SV_SEGMENTS; ++i) {
        uint32_t level;

        level = (pattern & (0x01U << (7 - i))) >> (7 - i);
        ESP_ERROR_CHECK(gpio_set_level(SV_DS, level));
        ESP_ERROR_CHECK(gpio_set_level(SV_SHCP, 0));
        ESP_ERROR_CHECK(gpio_set_level(SV_STOP, 1));
    }
    return;
}

// --- functions (static)
// =====
// Name : sv_fix
// Function : output from shift register to seven segment LED
// Parameters : none
// Return : none
// notes : none

// =====
static void sv_fix(void)
{
    ESP_ERROR_CHECK(gpio_set_level(SV_STOP, 0));
    ESP_ERROR_CHECK(gpio_set_level(SV_STOP, 1));
    return;
}
```

データを
7セグメントLED
に出力する関数