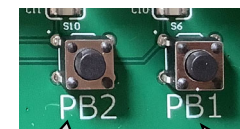
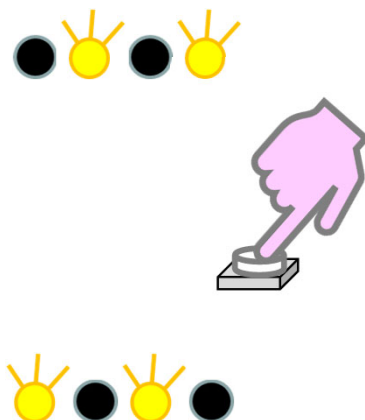


例題2 割込み処理

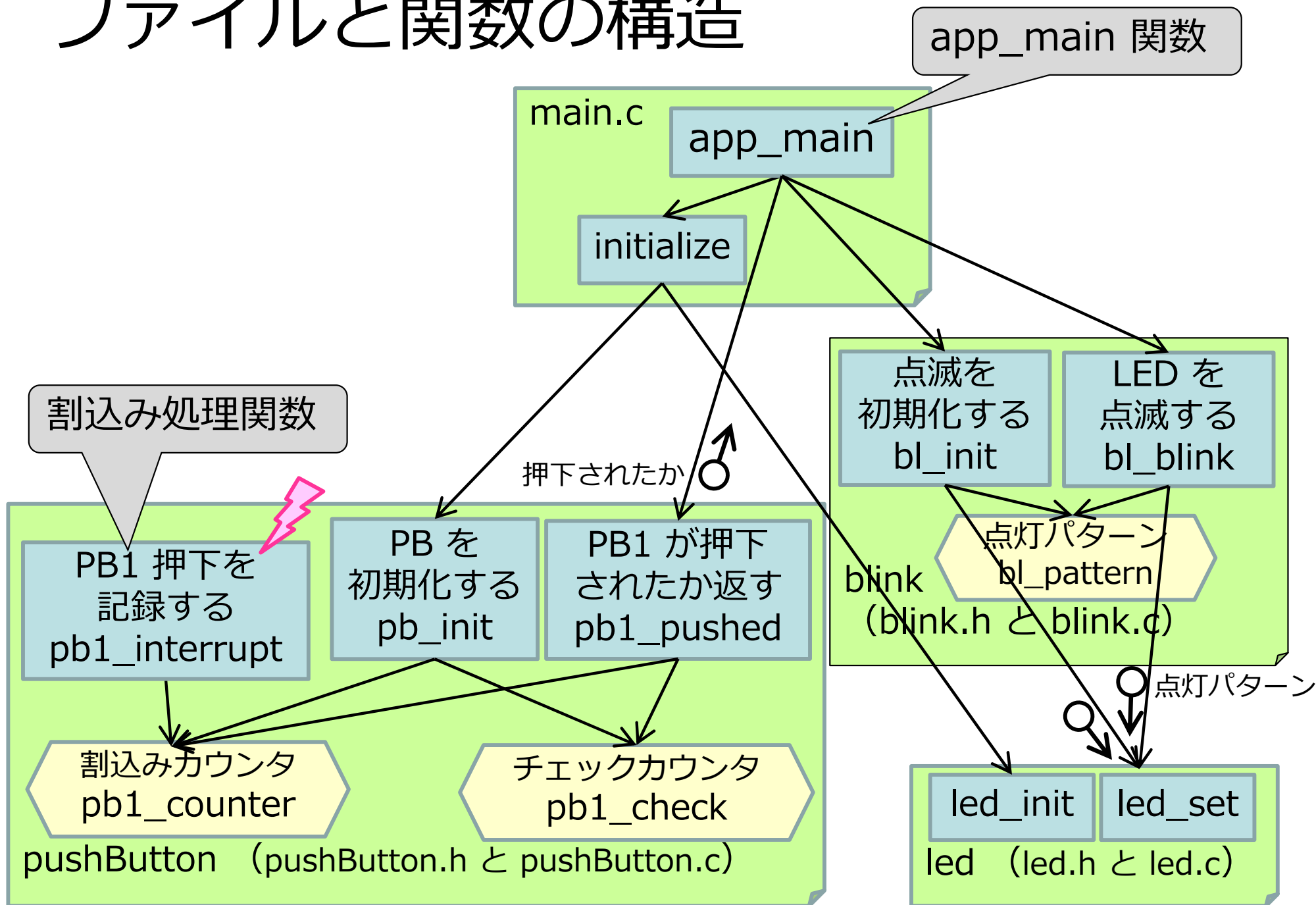
- PB1 が押下されたときに
LED のパターンを反転する



PB2

PB1

ファイルと関数の構造

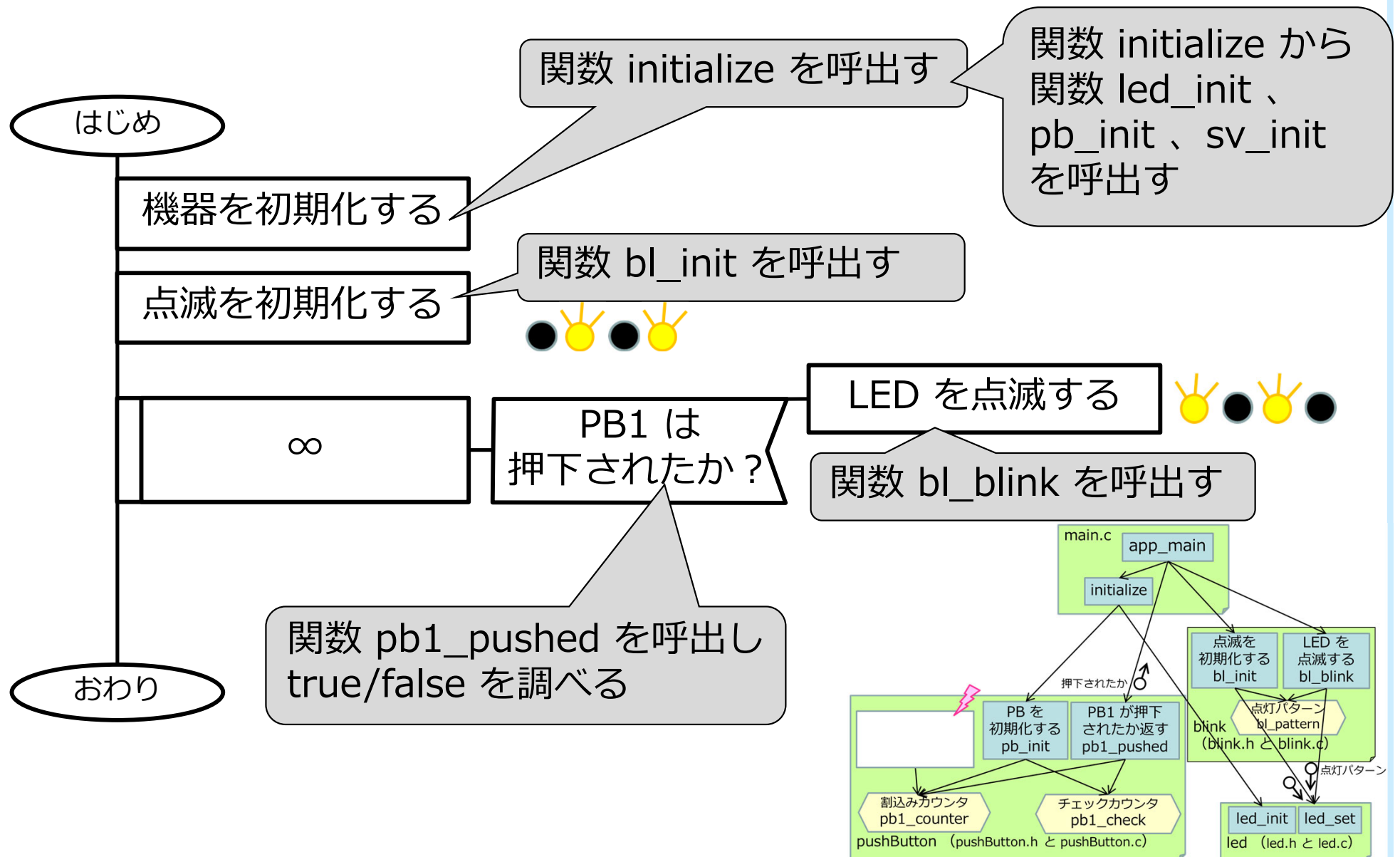


ファイルと関数の仕様

ファイル/責務	関数	機能	入力/引数	出力/返却値
main.c システム動作	app_main	動作させる	情報	なし
	initialize	初期化する	なし	なし
blink LED点滅	bl_init	点滅を 初期化する	なし	なし
	bl_blink	LED を 点滅する	なし	なし
led LED	led_init	LED を 初期化する	なし	なし
	led_set	LED を 点灯する	点灯 パターン	なし
pushButton プッシュボタン	pb_init	ブッシュボタン を初期化する	なし	なし
	pb1_pushed	PB1 が押下され たか返す	なし	true : 押下された false : 押下なし
	割り込み処理関数 pb1_interrupt	PB1 押下を 記録する	情報	なし

ヘッダファイル stdbool.h を
#include して bool 型の定数 true/false を使う

app_main 関数のアルゴリズム



ファイル main.c

■ ヘッダファイルの #include ファイル main.c 内で定義/使用する関数の プロトタイプ

```
// --- Header files (system)
#include <stdbool.h>
// --- Header files (project)
#include "led.h"
#include "sevenSegmentLed.h"
#include "pushButton.h"
#include "blink.h"
```

```
// --- prototypes (static)
static void initialize(void);
```

true/false を使用するため

ファイル pushButton.c で
定義されている関数
pb_init と pb1_pushed の
プロトタイプ

ファイル blink.c で
定義されている関数
bl_init と bl_blink の
プロトタイプ

ファイル main.c 内でのみ
定義/使用される関数の
プロトタイプ

ファイル main.c の関数

```
void app_main(void)
```

```
{
```

```
    initialize();
```

装置を初期化する

```
    bl_init();
```

点滅を初期化する

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

```
        if (pb1_pushed() != false) {
```

PB1 は押下されたか？

```
            bl_blink();
```

LED を点滅する

```
        }
```

```
    }
```

```
}
```

```
static void initialize(void)
```

```
{
```

```
    led_init();
```

LED を初期化する

```
    sv_init();
```

```
    pb_init();
```

```
    return;
```

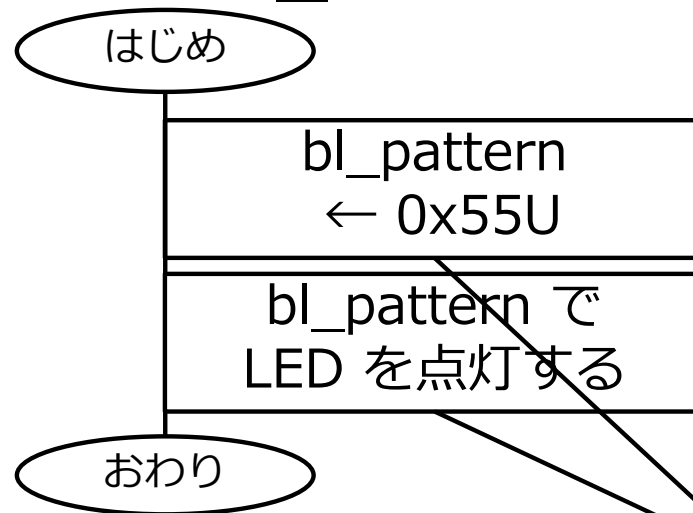
プッシュボタンを
初期化する

```
}
```

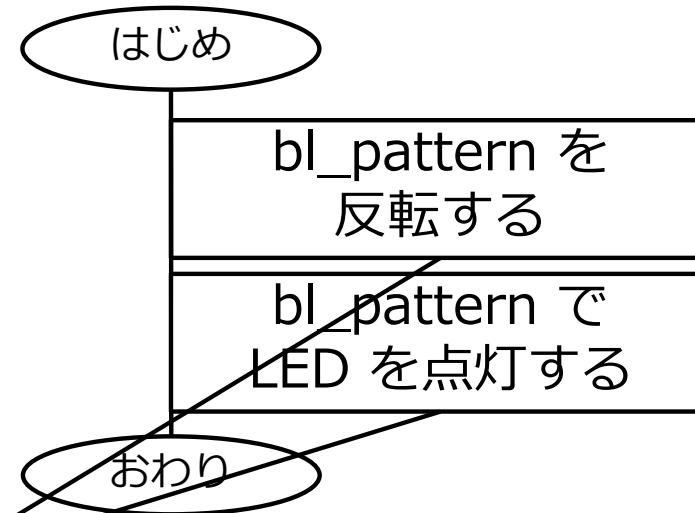
ファイル blink

■ 関数のアルゴリズム

関数 bl_init



関数 bl_blink



点灯パターン
`bl_pattern`

変数	意味
<code>bl_pattern</code>	点灯パターン

関数	機能	引数	返却値
<code>bl_init</code>	点滅を初期化する	なし	なし
<code>bl_blink</code>	LED を点滅する	なし	なし

ヘッダファイル blink.h

```
// --- prototypes (extern)  
extern void bl_init(void);  
extern void bl_blink(void);
```

ファイル blink.c で定義され
他のファイルで使われる関数
(他のファイルの関数から呼出される関数)
bl_init と bl_blink のプロトタイプ

ファイル blink.c

■ ヘッダファイルの #include

```
// --- Header files (project)
#include "led.h"
// --- Header of own file
#include "blink.h"
```

ファイル blink.c で使用する関数のプロトタイプ

ファイル blink.c で定義する関数のプロトタイプ

■ ファイル blink.c 内で使用するマクロ

```
// --- macros
#define BL_INIT      0x05U
#define LOWER_4BIT   0x0FU
```

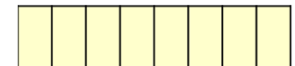
■ ファイル blink.c 内で使用するデータ

```
// --- data (static)
static unsigned char bl_pattern;
```

点灯パターン
bl_pattern

ファイル内で使用するデータ

bl_pattern



関数 bl_init と bl_blink の定義

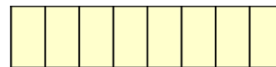
■ 関数 bl_init

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

■ 関数 bl_blink

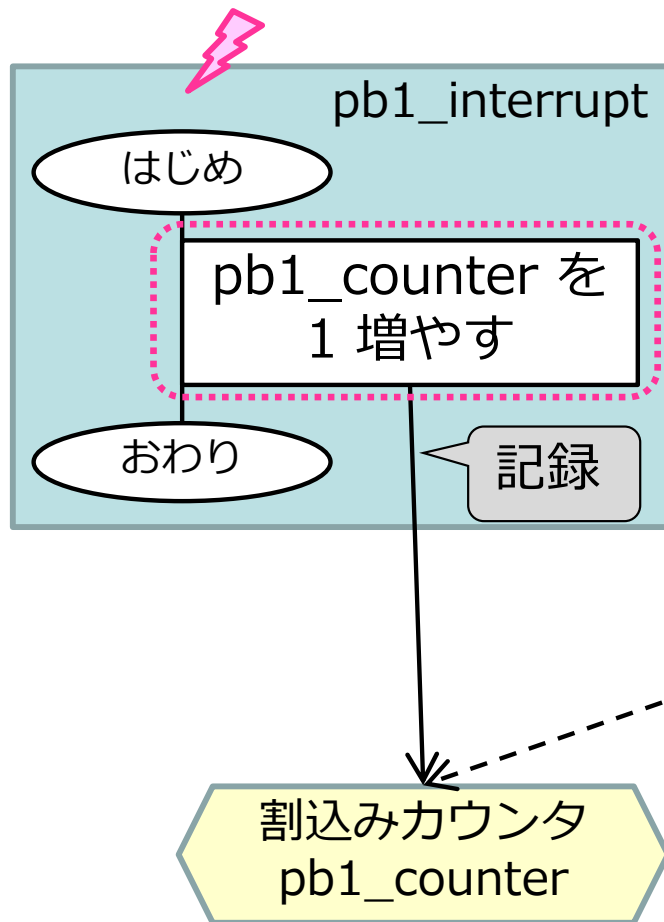
```
void bl_blink(void)
{
    bl_pattern = bl_pattern ^ LOWER_4BIT;
    led_set(bl_pattern);
    return;
}
```

点灯パターン
bl_pattern

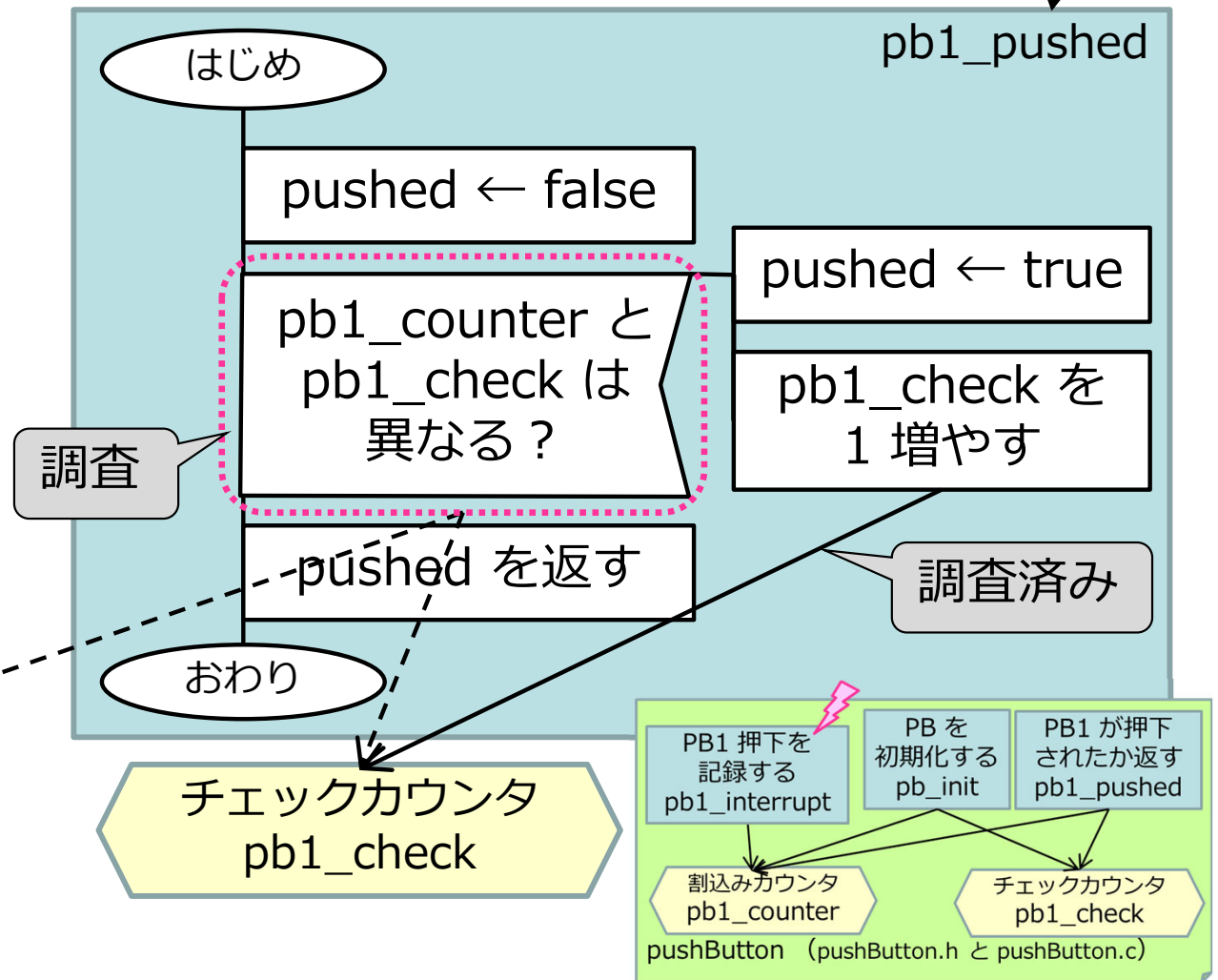


割り込み発生時の記録とその調査

■ 関数 pb1_interrupt 記録



■ 関数 pb1_pushed 調査



ヘッダファイル bushButton.h

```
// --- prototypes (extern)
extern void pb_init(void);
extern bool pb1_pushed(void);
extern bool pb2_pushed(void);
```

ファイル pushButton.c で定義され
他のファイルで使われる関数
(他のファイルの関数から呼出される関数)
のプロトタイプ

ファイル bushButton.c

■ ヘッダファイルの #include

```
// --- Header files (system)
#include <stdbool.h>
#include "driver/gpio.h"
#include "hal/gpio_types.h"
// --- Header of own file
#include "pushButton.h"
```

■ マクロの定義

```
// --- macros
#define PB1 GPIO_NUM_36
#define PB2 GPIO_NUM_39
```

■ データの定義

割込み発生を記録する変数

```
// --- data (static)
// counter that counts interrupt occurrence
static volatile unsigned char pb1_counter;
static volatile unsigned char pb2_counter;
```

割込み発生を調査する変数

```
// counter that counts checked interrupts
static unsigned char pb1_check;
static unsigned char pb2_check;
```

■ プロトタイプ

割込み処理関数

```
// --- prototypes (static)
static void pb1_interrupt(void *arg);
static void pb2_interrupt(void *arg);
```

記録とその調査のプログラム

■ 関数 pb1_interrupt

```
static void pb1_interrupt(void *arg)
{
    ++pb1_counter;
    return;
}
```

記録

割込みカウンタ
pb1_counter

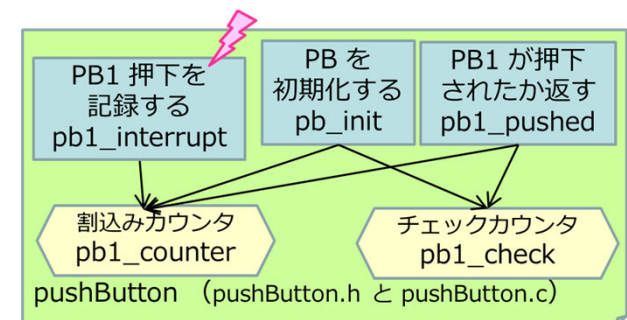
■ 関数 pb1_pushed

```
bool pb1_pushed(void)
{
    bool    pushed = false;
    if (pb1_counter != pb1_check) {
        ++pb1_check;
        pushed = true;
    }
    return pushed;
}
```



調査

調査済み

チェックカウンタ
pb1_check



補足：ふたつのカウンタの変化

タイミング	関数	割込みカウンタ pb1_counter	チェックカウンタ pb1_check
初期化時	pb_init	0	0
:		0	0
pb1_pushed 呼出し前	app_main	0	0
pb1_pushed return 時	pb1_pushed	0	0
:	返却値 false	0	0
 PB1 押下	pb1_interrupt	1	0
pb1_pushed 呼出し前	app_main	1	0
pb1_pushed return 時	pb1_pushed	1	1
:	返却値 true	1	1
 PB1 押下	pb1_interrupt	2	1
pb1_pushed 呼出し前	app_main	2	1
pb1_pushed return 時	pb1_pushed	2	2
:	返却値 true		

参考：割り込み処理関数の登録

■ プッシュボタンの初期化関数 pb_init

➤ プッシュボタン PB2 の記述省略

```
void pb_init(void)
{
    esp_rom_gpio_pad_select_gpio(PB1);
    ESP_ERROR_CHECK(gpio_set_direction(PB1, GPIO_MODE_INPUT));
    ESP_ERROR_CHECK(gpio_pullup_dis(PB1));
    ESP_ERROR_CHECK(gpio_pulldown_en(PB1));
    ESP_ERROR_CHECK(gpio_set_intr_type(PB1, GPIO_INTR_POSEDGE));
    // initialize interrupt
    ESP_ERROR_CHECK(gpio_install_isr_service(0));
    ESP_ERROR_CHECK(gpio_isr_handler_add(PB1, pb1_interrupt, NULL));

    pb1_counter = 0U;
    pb1_check = pb1_counter;
    return;
}
```

割り込み処理関数を登録