AKHIL CHERUKURI
CMPE 244
LAB: Interrupts and Binary Semaphores

Code can be found on: https://gitlab.com/akhilcherukuri/sjtwo-c/-/merge_requests/4/diffs

Note:

#define _____

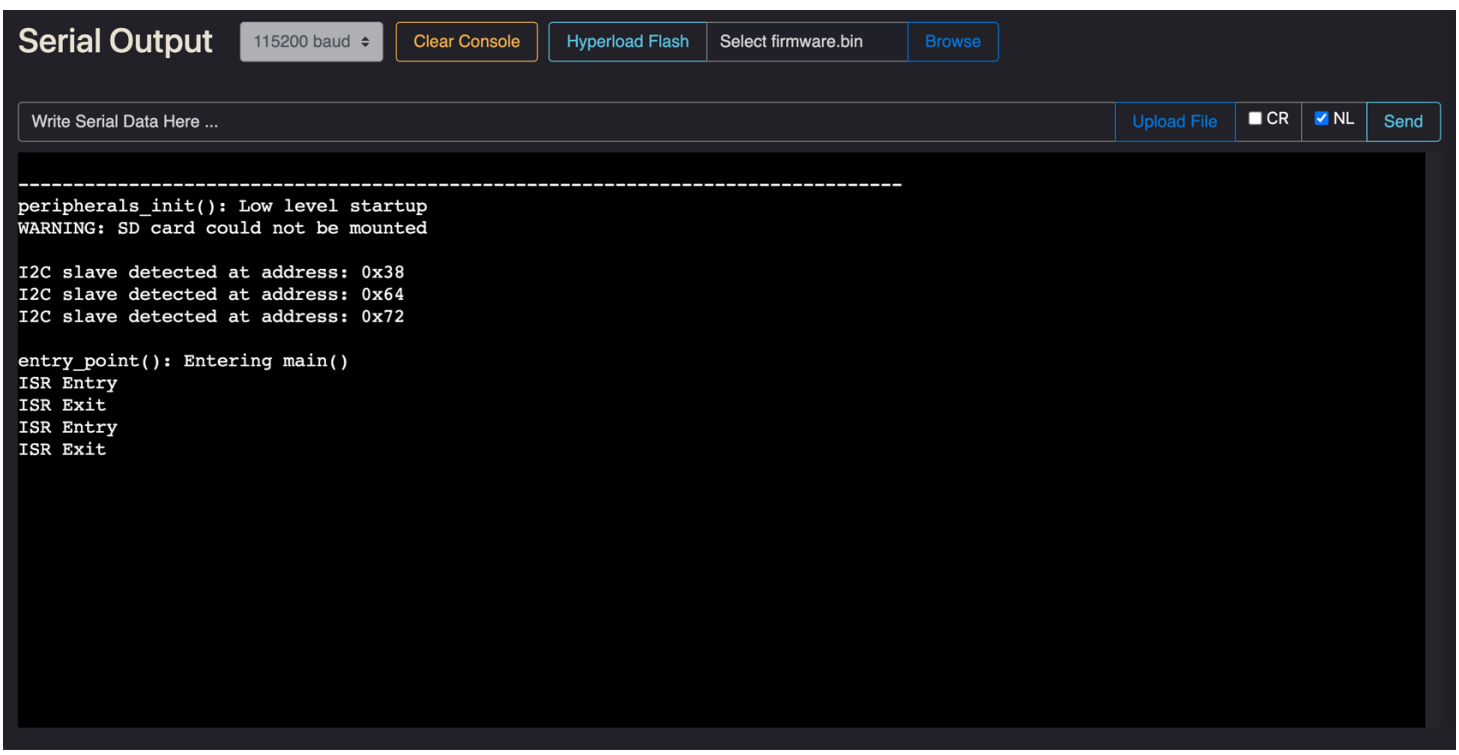
- Part_0 for Part 0
- Part_1 for Part 1
- Part_2 for Part 2
- extra_credit for Multiple Ports (Port 0 and 2)

configurable in main.c, gpio_isr.h, gpio_isr.c, interrupt_vector_table.c

---

PART 0:

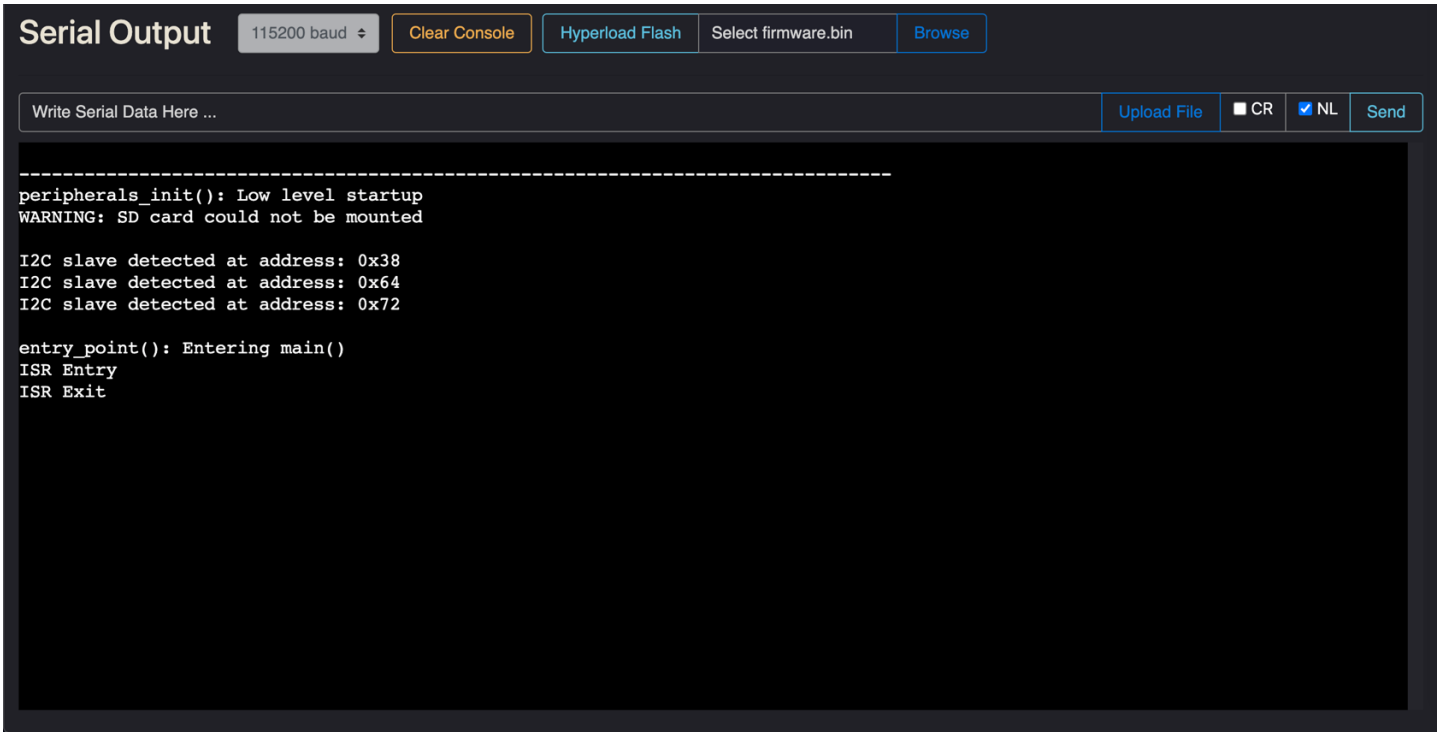
Simple Interrupt hijacked the interrupt vector at interrupt_vector_table.c



AKHIL CHERUKURI
CMPE 244
LAB: Interrupts and Binary Semaphores

PART 1:

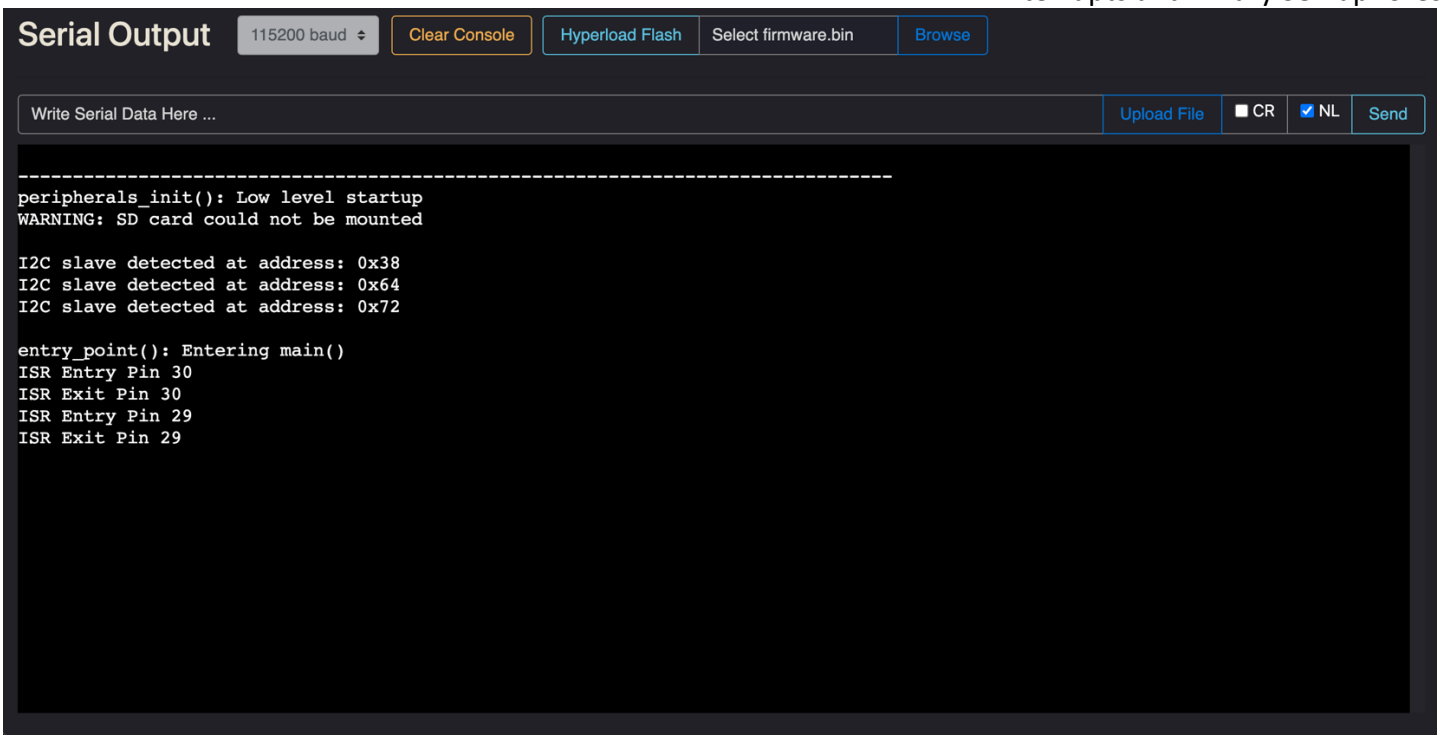
Interrupt with Binary Semaphore and using API form lpc_pheriphers.h



PART 2:

Supporting GPIO interrupts using function pointers.

Extra Credit – Multiple Ports 0 and 2



Main.c

```
#include <stdio.h>

#include "FreeRTOS.h"
#include "semphr.h"
#include "task.h"

#include "delay.h"
#include "gpio.h"
#include "gpio_isr.h"
#include "lpc40xx.h"
#include "lpc_peripherals.h"
#include "sj2_cli.h"

// ************************change to Part_0,Part_1,Part_2,extra_credit************************
#define extra_credit

// ************************change to extra_credit,Part_2************************
#ifdef extra_credit
static SemaphoreHandle_t switch_pressed_signal;
static SemaphoreHandle_t switch_pressed_signal2;

void pin30_isr(void);
void pin29_isr(void);
void pin30_interrupt(void *p);
void pin29_interrupt(void *p);
```

```
static gpio_s led1 = {1, 24};
static gpio_s led2 = {1, 18};

void pin30_isr(void) {
  fprintf(stderr, "ISR Entry Pin 30 \n");
  xSemaphoreGiveFromISR(switch_pressed_signal, NULL);
  fprintf(stderr, "ISR Exit Pin 30 \n");
}

void pin29_isr(void) {
  fprintf(stderr, "ISR Entry Pin 29 \n");
  xSemaphoreGiveFromISR(switch_pressed_signal2, NULL);
  fprintf(stderr, "ISR Exit Pin 29 \n");
}

void configure_your_gpio_interrupt() {
  gpio__construct_as_input(GPIO__PORT_0, 30);
  gpio__construct_as_input(GPIO__PORT_0, 29);
  gpio__construct_as_output(GPIO__PORT_1, 24);
  gpio__construct_as_output(GPIO__PORT_1, 18);
}

void pin30_interrupt(void *p) {
  while (true) {
    if (xSemaphoreTake(switch_pressed_signal, 1000)) {
      gpio__set(led1);
      vTaskDelay(500);
      gpio__reset(led1);
      vTaskDelay(500);
    }
  }
}

void pin29_interrupt(void *p) {
  while (true) {
    if (xSemaphoreTake(switch_pressed_signal2, 1000)) {
      gpio__set(led2);
      vTaskDelay(500);
      gpio__reset(led2);
      vTaskDelay(500);
    }
  }
}

#endif

#ifdef Part_1
```

```
static SemaphoreHandle_t switch_pressed_signal;
void gpio_interrupt(void);
void sleep_on_sem_task(void *p);
void configure_your_gpio_interrupt(void);
void clear_gpio_interrupt(void);

static gpio_s sw2 = {0, 30};
static gpio_s led_sw2 = {1, 24};

void configure_your_gpio_interrupt() {
  gpio__set_as_input(sw2);
  gpio__set_as_output(led_sw2);
  gpio__enable_pull_down_resistors(sw2);

  LPC_GPIOINT->IO0IntEnF |= (1 << 30);
}

void clear_gpio_interrupt() { LPC_GPIOINT->IO0IntClr |= (1 << 30); }

void gpio_interrupt(void) {
  fprintf(stderr, "ISR Entry \n");
  xSemaphoreGiveFromISR(switch_pressed_signal, NULL);
  clear_gpio_interrupt();
  fprintf(stderr, "ISR Exit \n");
}

void sleep_on_sem_task(void *p) {
  while (1) {
    if (xSemaphoreTake(switch_pressed_signal, portMAX_DELAY)) {
      delay__ms(250);
      gpio__set(led_sw2);
      delay__ms(250);
      gpio__reset(led_sw2);
    }
  }
}
#endif

#ifdef Part_0
void gpio_interrupt(void) {
  fprintf(stderr, "ISR Entry \n");
  LPC_GPIOINT->IO0IntClr |= (1 << 30);
  fprintf(stderr, "ISR Exit \n");
}
#endif

int main(void) {
```

```
#ifdef Part_0
  static gpio_s sw2 = {0, 30};
  static gpio_s led_sw2 = {1, 24};
  gpio__set_as_input(sw2);
  gpio__set_as_output(led_sw2);
  gpio__enable_pull_down_resistors(sw2);

  LPC_GPIOINT->IO0IntEnF |= (1 << 30);

  NVIC_EnableIRQ(GPIO_IRQn);

  while (1) {
    delay__ms(250);
    gpio__set(led_sw2);
    delay__ms(250);
    gpio__reset(led_sw2);
  }
#endif

#ifdef Part_1
  switch_pressed_signal = xSemaphoreCreateBinary();
  lpc_peripheral__enable_interrupt(LPC_PERIPHERAL__GPIO, gpio_interrupt, "gpio_interrupt");

  configure_your_gpio_interrupt();
  NVIC_EnableIRQ(GPIO_IRQn);

  xTaskCreate(sleep_on_sem_task, "sem", (512U * 4) / sizeof(void *), NULL, PRIORITY_LOW, NULL);
  vTaskStartScheduler();
#endif

#ifdef Part_2
  switch_pressed_signal = xSemaphoreCreateBinary();
  switch_pressed_signal2 = xSemaphoreCreateBinary();

  lpc_peripheral__enable_interrupt(LPC_PERIPHERAL__GPIO, gpio0__interrupt_dispatcher, "gpio_interrupt");
  configure_your_gpio_interrupt();
  NVIC_EnableIRQ(GPIO_IRQn);

  gpio0__attach_interrupt(30, GPIO_INTR__RISING_EDGE, pin30_isr);
  gpio0__attach_interrupt(29, GPIO_INTR__FALLING_EDGE, pin29_isr);

  xTaskCreate(pin30_interrupt, "pin30_interrupt", (512U * 4) / sizeof(void *), NULL, PRIORITY_LOW, NULL);
  xTaskCreate(pin29_interrupt, "pin29_interrupt", (512U * 4) / sizeof(void *), NULL, PRIORITY_LOW, NULL);
  vTaskStartScheduler();
#endif

#ifdef extra_credit
  switch_pressed_signal = xSemaphoreCreateBinary();
```

```
  switch_pressed_signal2 = xSemaphoreCreateBinary();

  lpc_peripheral__enable_interrupt(LPC_PERIPHERAL__GPIO, gpio_ports__interrupt_dispatcher, "gpio_interrupt");
  configure_your_gpio_interrupt();
  NVIC_EnableIRQ(GPIO_IRQn);

  gpio_ports__attach_interrupt(0, 30, GPIO_INTR__RISING_EDGE, pin30_isr);
  gpio_ports__attach_interrupt(0, 29, GPIO_INTR__FALLING_EDGE, pin29_isr);

  xTaskCreate(pin30_interrupt, "pin30_interrupt", (512U * 4) / sizeof(void *), NULL, PRIORITY_LOW, NULL);
  xTaskCreate(pin29_interrupt, "pin29_interrupt", (512U * 4) / sizeof(void *), NULL, PRIORITY_LOW, NULL);
  vTaskStartScheduler();
#endif

  return 0;
}
```

gpio_isr.h

```
#pragma once

#include "stdio.h"
#include "stdlib.h"

// ************************Part_2 for only Port 0************************
// ************************extra_credit for Ports 0,2************************
#define extra_credit

#ifdef Part_2
typedef enum {
  GPIO_INTR__FALLING_EDGE,
  GPIO_INTR__RISING_EDGE,
} gpio_interrupt_e;

// Function pointer type (demonstrated later in the code sample)
typedef void (*function_pointer_t)(void);

// Allow the user to attach their callbacks
void gpio0__attach_interrupt(uint32_t pin, gpio_interrupt_e interrupt_type, function_pointer_t callback);

// Our main() should configure interrupts to invoke this dispatcher where we will invoke user attached callbacks
// You can hijack 'interrupt_vector_table.c' or use API at lpc_peripherals.h
void gpio0__interrupt_dispatcher(void);
#endif

#ifdef extra_credit
```

```
typedef enum {
  GPIO_INTR__FALLING_EDGE,
  GPIO_INTR__RISING_EDGE,
} gpio_interrupt_e;

// Function pointer type (demonstrated later in the code sample)
typedef void (*function_pointer_t)(void);

// Allow the user to attach their callbacks
void gpio_ports__attach_interrupt(uint8_t port, uint32_t pin, gpio_interrupt_e interrupt_type,
                                  function_pointer_t callback);

// Our main() should configure interrupts to invoke this dispatcher where we will invoke user attached callbacks
// You can hijack 'interrupt_vector_table.c' or use API at lpc_peripherals.h
void gpio_ports__interrupt_dispatcher(void);

#endif
```

gpio-isr.c:

```
#include "gpio_isr.h"
#include "lpc40xx.h"

// ************************Part_2 for only Port 0************************
// ************************extra_credit for Ports 0,2************************
#define extra_credit

#ifdef Part_2
// Note: You may want another separate array for falling vs. rising edge callbacks
static function_pointer_t gpio0_falling_edge_callbacks[32];
static function_pointer_t gpio0_rising_edge_callbacks[32];
#endif

#ifdef extra_credit
// 0 for port 0 and 1 for port 2
static function_pointer_t gpios_falling_edge_callbacks[2][32];
static function_pointer_t gpios_rising_edge_callbacks[2][32];
#endif

#ifdef Part_2
void gpio0__attach_interrupt(uint32_t pin, gpio_interrupt_e interrupt_type, function_pointer_t callback) {
  // 1) Store the callback based on the pin at gpio0_callbacks
  // 2) Configure GPIO 0 pin for rising or falling edge
  if (interrupt_type == GPIO_INTR__FALLING_EDGE) {
    gpio0_falling_edge_callbacks[pin] = callback;
```

```
    LPC_GPIOINT->IO0IntEnF |= (1U << pin);
  } else {
    gpio0_rising_edge_callbacks[pin] = callback;
    LPC_GPIOINT->IO0IntEnR |= (1U << pin);
  }
}

static void clear_pin_interrupt(uint8_t pin) { LPC_GPIOINT->IO0IntClr |= (1U << pin); }

// We wrote some of the implementation for you
void gpio0__interrupt_dispatcher(void) {
  // Check which pin generated the interrupt
  uint8_t pin_that_generated_interrupt = 0;

  while ((LPC_GPIOINT->IO0IntStatF >> pin_that_generated_interrupt) && (pin_that_generated_interrupt <= 31)) {
    if (LPC_GPIOINT->IO0IntStatF >> pin_that_generated_interrupt & 1) {
      if (gpio0_falling_edge_callbacks[pin_that_generated_interrupt]) {
        function_pointer_t attached_user_handler = gpio0_falling_edge_callbacks[pin_that_generated_interrupt];
        attached_user_handler();
      }
      clear_pin_interrupt(pin_that_generated_interrupt);
    }
    pin_that_generated_interrupt++;
  }
  while ((LPC_GPIOINT->IO0IntStatR >> pin_that_generated_interrupt) && (pin_that_generated_interrupt <= 31)) {
    if (LPC_GPIOINT->IO0IntStatR >> pin_that_generated_interrupt & 1) {
      if (gpio0_rising_edge_callbacks[pin_that_generated_interrupt]) {
        function_pointer_t attached_user_handler = gpio0_rising_edge_callbacks[pin_that_generated_interrupt];
        attached_user_handler();
      }
      clear_pin_interrupt(pin_that_generated_interrupt);
    }
    pin_that_generated_interrupt++;
  }
}

#endif

#ifdef extra_credit

void gpio_ports__attach_interrupt(uint8_t port, uint32_t pin, gpio_interrupt_e interrupt_type,
                                  function_pointer_t callback) {
  if (port == 0) {
    if (interrupt_type == GPIO_INTR__FALLING_EDGE) {
      gpios_falling_edge_callbacks[0][pin] = callback;
      LPC_GPIOINT->IO0IntEnF |= (1U << pin);
    } else {
      gpios_rising_edge_callbacks[0][pin] = callback;
```

```
      LPC_GPIOINT->IO0IntEnR |= (1U << pin);
    }
  } else if (port == 2) {
    if (interrupt_type == GPIO_INTR__FALLING_EDGE) {
      gpios_falling_edge_callbacks[1][pin] = callback;
      LPC_GPIOINT->IO2IntEnF |= (1U << pin);
    } else {
      gpios_rising_edge_callbacks[1][pin] = callback;
      LPC_GPIOINT->IO2IntEnR |= (1U << pin);
    }
  }
}

static void clear_pins_interrupt(uint8_t port, uint8_t pin) {
  if (port == 0) {
    LPC_GPIOINT->IO0IntClr |= (1U << pin);
  } else if (port == 2) {
    LPC_GPIOINT->IO2IntClr |= (1U << pin);
  }
}

void gpio_ports__interrupt_dispatcher(void) {
  // Check which pin generated the interrupt
  uint8_t pin_that_generated_interrupt = 0;

  while ((LPC_GPIOINT->IO0IntStatF >> pin_that_generated_interrupt) && (pin_that_generated_interrupt <= 31)) {
    if (LPC_GPIOINT->IO0IntStatF >> pin_that_generated_interrupt & 1) {
      if (gpios_falling_edge_callbacks[0][pin_that_generated_interrupt]) {
        function_pointer_t attached_user_handler =
gpios_falling_edge_callbacks[0][pin_that_generated_interrupt];
        attached_user_handler();
      }
      clear_pins_interrupt(0, pin_that_generated_interrupt);
    }
    pin_that_generated_interrupt++;
  }
  while ((LPC_GPIOINT->IO0IntStatR >> pin_that_generated_interrupt) && (pin_that_generated_interrupt <= 31)) {
    if (LPC_GPIOINT->IO0IntStatR >> pin_that_generated_interrupt & 1) {
      if (gpios_rising_edge_callbacks[0][pin_that_generated_interrupt]) {
        function_pointer_t attached_user_handler = gpios_rising_edge_callbacks[0][pin_that_generated_interrupt];
        attached_user_handler();
      }
      clear_pins_interrupt(0, pin_that_generated_interrupt);
    }
    pin_that_generated_interrupt++;
  }
  while ((LPC_GPIOINT->IO2IntStatF >> pin_that_generated_interrupt) && (pin_that_generated_interrupt <= 31)) {
    if (LPC_GPIOINT->IO2IntStatF >> pin_that_generated_interrupt & 1) {
```

```
      if (gpios_falling_edge_callbacks[1][pin_that_generated_interrupt]) {
        function_pointer_t attached_user_handler =
gpios_falling_edge_callbacks[1][pin_that_generated_interrupt];
        attached_user_handler();
      }
      clear_pins_interrupt(2, pin_that_generated_interrupt);
    }
    pin_that_generated_interrupt++;
  }
  while ((LPC_GPIOINT->IO2IntStatF >> pin_that_generated_interrupt) && (pin_that_generated_interrupt <= 31)) {
    if (LPC_GPIOINT->IO2IntStatF >> pin_that_generated_interrupt & 1) {
      if (gpios_falling_edge_callbacks[1][pin_that_generated_interrupt]) {
        function_pointer_t attached_user_handler =
gpios_falling_edge_callbacks[1][pin_that_generated_interrupt];
        attached_user_handler();
      }
      clear_pins_interrupt(2, pin_that_generated_interrupt);
    }
    pin_that_generated_interrupt++;
  }
}

#endif
```