ADXL345 Interrupt Configuration and ESP32 Integration Guide

1. Configuring INT1/INT2 Pins on the ADXL345

The ADXL345 accelerometer features two interrupt output pins, INT1 and INT2, which can be configured to signal various events like motion detection or free-fall.

- Interrupt Output Pins: Both INT1 and INT2 are push-pull, low-impedance pins. By default, they are configured as active high but can be set to active low by modifying the INT_INVERT bit in the DATA_FORMAT register (Address 0x31).

- Mapping Interrupts: Use the INT_MAP register (Address 0x2F) to assign specific interrupt functions to either INT1 or INT2. Setting a bit to 0 maps the corresponding interrupt to INT1; setting it to 1 maps it to INT2.

- Enabling Interrupts: The INT_ENABLE register (Address 0x2E) is used to enable specific interrupt functions. Setting a bit to 1 enables the corresponding interrupt.

Example to enable free-fall detection and map it to INT1:

```
// Enable free-fall interrupt
writeTo(ADXL345_ADDRESS, 0x2E, 0x04); // INT_ENABLE register

// Map free-fall interrupt to INT1
writeTo(ADXL345_ADDRESS, 0x2F, 0x00); // INT_MAP register
```

2. Enabling Motion or Free-Fall Detection Using Internal Registers

The ADXL345 provides built-in features for detecting motion and free-fall events, reducing the need for complex algorithms on the host processor.

- Free-Fall Detection:
  - THRESH_FF (0x28): Sets the free-fall threshold.
  - TIME_FF (0x29): Sets the time duration.

- Motion Detection:
  - THRESH_ACT (0x24): Sets the activity threshold.
  - ACT_INACT_CTL (0x27): Axis and coupling config.

Example to configure free-fall detection:

```
// Set free-fall threshold to 300 mg
writeTo(ADXL345_ADDRESS, 0x28, 0x05); // THRESH_FF register

// Set free-fall time to 100 ms
writeTo(ADXL345_ADDRESS, 0x29, 0x14); // TIME_FF register

// Enable free-fall interrupt
writeTo(ADXL345_ADDRESS, 0x2E, 0x04); // INT_ENABLE register

// Map free-fall interrupt to INT1
writeTo(ADXL345_ADDRESS, 0x2F, 0x00); // INT_MAP register
```

3. Interfacing the Interrupt Pin with an ESP32 GPIO

To interface the ADXL345's interrupt pin with the ESP32:

1. Hardware Connection:
   - Connect INT1 or INT2 to an ESP32 GPIO.
   - Ensure the GPIO supports interrupts.

2. ESP32 GPIO Configuration:
   - Configure as input with the appropriate interrupt type.

Example for GPIO 23:

```
#define ADXL345_INT_GPIO GPIO_NUM_23

gpio_config_t io_conf = {
   .intr_type = GPIO_INTR_POSEDGE,
   .mode = GPIO_MODE_INPUT,
   .pin_bit_mask = (1ULL << ADXL345_INT_GPIO),
   .pull_down_en = GPIO_PULLDOWN_DISABLE,
   .pull_up_en = GPIO_PULLUP_ENABLE,
};
gpio_config(&io_conf);
```

4. Handling the Interrupt in ESP-IDF

In the ESP-IDF environment, handling GPIO interrupts involves setting up an ISR and associating it with the GPIO pin.

Steps:
1. Install ISR Service

2. Attach ISR Handler
3. Define ISR Handler

Example implementation:

```
#include "driver/gpio.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"

#define ADXL345_INT_GPIO GPIO_NUM_23

static void IRAM_ATTR adxl345_isr_handler(void* arg) {
  // Minimal ISR: set a flag or notify a task
}

void app_main(void) {
  gpio_config_t io_conf = {
    .intr_type = GPIO_INTR_POSEDGE,
    .mode = GPIO_MODE_INPUT,
    .pin_bit_mask = (1ULL << ADXL345_INT_GPIO),
    .pull_down_en = GPIO_PULLDOWN_DISABLE,
    .pull_up_en = GPIO_PULLUP_ENABLE,
  };
  gpio_config(&io_conf);

  gpio_install_isr_service(ESP_INTR_FLAG_DEFAULT);
  gpio_isr_handler_add(ADXL345_INT_GPIO, adxl345_isr_handler,
(void*) ADXL345_INT_GPIO);

  while (1) {
    vTaskDelay(pdMS_TO_TICKS(1000));
  }
}
```