Работа с энкодером. Указатель на функцию

В реальности энкодер используется в более сложных ситуациях, и придётся менять минимум три переменных: яркость дисплея в режиме отображения времени; часы в режиме настройки часов; и минуты в режиме настройки минут. Вычитая из текущего состояния регистра `CNT` его предыдущее значение мы можем узнать скорость вращения и направление. Но показания счётчика зациклены. Рассмотрим крайние случаи. Пусть счётчик хранит значение `MAX_COUNT_VALUE - 1`, тогда при повороте по часовой стрелке, он прибавляя `2` перегрузит счётчик и вместо ожидаемых `129` в нём будет записано `1`. Другими словами нам нужно, чтобы:

$$127 + 1 \Rightarrow 2,$$
$$1 - 127 \Rightarrow -2$$

За один оборот вала должно произойти 11 щелчков, согласно документации на энкодер. Человек вряд ли сможет сделать несколько оборотов менее чем, скажем, за 0,25 с, следовательно алгоритм подсчёта разницы можно сделать очень «топорным»: если модуль разницы текущего состояния и предыдущего больше половины от `MAX_COUNT_VALUE`, то мы, условно (зависит от подключения) крутили вправо, в противном случае влево. Значение счётчика меняется на число кратное двум, это не удобно, поэтому конечный результат нужно поделить на 2.

```
int16_t calc_diff(uint16_t curr, uint16_t prev) {
    int16_t diff = curr - prev;
    if (((diff < 0 ? -diff : diff)) > (MAX_COUNT_VALUE  >> 1)) { // abs(diff) > 64
        diff = (diff < 0) ? diff + MAX_COUNT_VALUE : diff - MAX_COUNT_VALUE;
    }
    return (diff >> 1); // devision by 2
}
```

Подсчитать разницу и как-то обработать её (если она отличная от нуля) можно в главном цикле.

```
while(1) {
    // ...
    encoder_previous_value = encoder_current_value;
    encoder_current_value = encoder_get_value();
    encoder_diff = calc_diff(encoder_current_value, encoder_previous_value);
    if (encoder_diff != 0) {
        // do some action here
    }
    // ...
}
```

Но это не лучшее решение, ведь время реакции системы непредсказуемо. Можно настроить прерывание по изменению счётчика.

```
void encoder_init(void) {
    // Инициализируем порты и таймер
    TIM3->DIER |= TIM_DIER_UIE;
    NVIC_EnableIRQ(TIM3_IRQn);
    // Включаем таймер
}
```

Это лучше с точки зрения отзывчивости системы, но при быстрой прокрутке прерывания будут идти одно за другим подтормаживая всю остальную систему, что уже не так хорошо. Плюсом код обработки счётчика придётся вынести в `stm32f10x_it.c`, что не очень удобно.

Лучше всего настроить ещё один таймер, который с некоторой периодичностью, например с частотой 4 - 10 Гц (0,25 - 0,1 с), будет вызывать прерывание и как-то реагировать на изменение счётчика. Накладные расходы будут постоянными, но не такими большими, чтобы это как-то сильно сказалось на всю систему. Для удобства заведём структуру.

```
// encoder.h
typedef struct {
    uint16_t current_value;
    uint16_t previous_value;
    int16_t diff;
} ENCODER_t;

// encoder.c
volatile ENCODER_t encoder = {
    .current_value  = 0,
    .previous_value = 0,
    .diff           = 0,
};
```

Добавим в функцию `encoder_init()` настройку таймера, например `TIM4`, который мы использовали ранее.

```
void encoder_init(void) {
    // ...
    // Включаем тактирование таймера TIM4
    RCC->APB1ENR |= RCC_APB1ENR_TIM4EN;

    TIM4->PSC = (SystemCoreClock / 2) / 1000 - 1; // 63999
    TIM4->ARR = 250 - 1;                          // 250 мс
    TIM4->DIER |= TIM_DIER_UIE;                   // разрешаем прерывания
    NVIC_EnableIRQ(TIM4_IRQn);                    // глобально разрешаем прерывание

    TIM4->CR1 |= TIM_CR1_CEN;                     // запускаем таймер
}
```

И сам обработчик прерывания.

```
// encoder.c
void encoder_update(void) {
    encoder.previous_value = encoder.current_value;
    encoder.current_value = encoder_get_value();
    encoder.diff = calc_diff(encoder.current_value, encoder.previous_value);
    if (encoder.diff != 0) {
        // do some action here
    }
}

// stm32f10x_it.h
#include "encoder.h"
```

```
// stm32f10x_it.c
void TIM3_IRQHandler(void) {
    encoder_update();
    TIM4->SR &= ~TIM_SR_UIF;
}
```

Чтобы ещё больше развязать логику работы ДУП от действия которое он выполняет (ведь действий может быть несколько, в зависимости от состояния), можно в нашу структуру добавить указатель на функцию и подменять её в зависимости от состояния, например так:

```
// encoder.h
typedef void (*callback)(int16_t);

typedef struct {
    // ...
    callback action;
} ENCODER_t;

void encoder_set_action(callback new_action);

// encoder.c
volatile ENCODER_t encoder = {
        // ...
        .action = 0,
};

void encoder_set_action(callback new_action) {
    // turn off interrupt
    encoder.action = new_action;
    // turn on interrupt
}
```

Оставлять указатель `action` равным `0` **плохая идея**. Переделаем немного функцию инициализации и принудим пользователя модулем указывать нужную функцию.

```
void encoder_init(callback action) {
    // ...
    encoder_set_action(action);
}
```

Не забудьте поправить прототип функции в заголовочном файле.

Теперь дополним `encoder_update()` и перепишем главный цикл.

```
void encoder_update(void) {
    // ...
    if (encoder.diff != 0) {
        (*(encoder.action))(encoder.diff);
    }
}

// main.c
uint8_t value = 0;
```

```
int main(void) {
    mcu_init(); // --> encoder_init(change_var);

    while (1) {
        led_toggle();
        delay(value * 10000);
    }
}

void change_var(int16_t diff) {
    value += diff;
}
```

> Если вы знакомы с C++, то наверняка заметили — здесь возможностями языка Си реализован «класс». Данная конструкция немного «уродливая», так как Си изначально не объекто-ориентированный язык программирования. В случае, если вы находите привязку функций к некоторому сущности интересной концепцией, то имеет смысл перейти на ООП. Книга «**C++ для встраиваемых систем**» (**в данный момент в состоянии написания**) посвящена именно этим вопросам.

Теперь, при повороте энкодера в прерывании вычисляться разность текущего и предыдущего состояния счётчика. Если разность отлична от нуля, то в прерывании будет вызвана функция `change_var()`. Подменяя указатель, можно добиться разной реакции на поворот энкодера.

Код урока можно найти на github: **CMSIS**.

Назад | **Оглавление** | **Дальше**