32 bity jak najprościej – STM32F0 (6)

Obsługa pól dotykowych

Wybrane modele mikrokontrolerów serii STM32F0 są wyposażone w moduł sprzętowy umożliwiający współpracę z polami dotykowymi. We współczesnych systemach mikrokomputerowych taki interfejs użytkownika jest atrakcyjną alternatywą dla tradycyjnych przycisków, oferując w porównaniu z nimi większą niezawodność, odporność mechaniczną i niższy koszt realizacji. W kolejnym odcinku serii przedstawiony zostanie przykład nieblokującej obsługi pól dotykowych zrealizowany na płytce z serii Discovery z mikrokontrolerem STM32F072.

Realizowane w mikrokontrolerach sterowniki obsługujące pola dotykowe działają na zasadzie pomiaru pojemności pomiędzy elektrodą i masą układu. Rozwiązania poszczególnych producentów różnią się metodą pomiaru pojemności i stopniem jej automatyzacji. W mikrokontrolerach serii STM32F0 zastosowano metodę transferu ładunku – pomiar pojemności elektrody dotykowej jest realizowany poprzez wielokrotne ładowanie pojemności tworzonej przez elektrodę dotykową i zliczanie cykli transferu ładunku z elektrody dotykowej do kondensatora gromadzącego ładunek potrzebnych do naładowania kondensatora do określonego napięcia.

Elektryczne aspekty projektowania pól dotykowych zostały szczegółowo omówione w notach aplikacyjnych AN2869 i AN4312, dostępnych w serwisie internetowym firmy ST Microelectronics. Projektując urządzenie z interfejsem dotykowym należy zapoznać się szczegółowo z zasadami projektowania pól dotykowych i prowadzenia połączeń na płytce drukowanej, gdyż mają one decydujące znaczenie dla czułości i poprawności działania pól.

Listing 1 – plik stm32f0discovery.h

```
/*
    STM32F0DISCOVERY, STM32F0308-DISCO & STM32F072B-
DISCO board defs
    gbm 05'2014
*/

#include "stm32f0yy.h"
#include "stm32futil.h"
//================================================
// STM32F0DISCOVERY board conections
#define BUTTON_PORT    GPIOA
#define BUTTON_BIT     0
#define LED_PORT         GPIOC
// LEDs - F0, F0308
#define BLUE_LED_BIT   8
#define GREEN_LED_BIT  9
#define BLUE_LED_PWM   TIM3->CCR3
#define GREEN_LED_PWM  TIM3->CCR4
// LEDs - F072B
#define LED_U_BIT    6
#define LED_D_BIT    7
#define LED_L_BIT    8
#define LED_R_BIT    9
#define LED_U_PWM    TIM3->CCR1
#define LED_D_PWM    TIM3->CCR2
#define LED_L_PWM    TIM3->CCR3
#define LED_R_PWM    TIM3->CCR4
// F072B touch
#define TSC_CAPS   (1 << 10 | 1 << 7 | 1 << 3)
#define TSC_KEYS   (1 << 9 | 1 << 6 | 1 << 2)
```

Interfejs dotykowy w STM32F0

Sterownik interfejsu dotykowego TSC zaimplementowany w mikrokontrolerach STM32F0 może obsłużyć do 24 pól w 8 grupach. Każda grupa ma do czterech wyprowadzeń zewnętrznych, z których jedno (dowolnie wybrane przez projektanta sprzętu) służy do podłączenia kondensatora gromadzącego ładunek, a pozostałe – do podłączenia elektrod dotykowych. Całkowita liczba pól obsługiwanych przez poszczególne modele mikrokontrolerów zależy od dostępności linii portów w obudowie mikrokontrolera. W jednym cyklu pomiarowym sterownik może zmierzyć pojemność jednego pola w każdej grupie. Liczba używanych grup zależy od decyzji projektanta na przykład, jeśli w urządzeniu są trzy pola, to można użyć jednej grupy (co skutkuje koniecznością sekwencyjnego testowania stanu poszczególnych pól, ale za to wymaga użycia tylko jednego kondensatora) lub trzech grup (dzięki czemu będzie możliwe testowanie stanu pól w jednym cyklu pomiarowym, ale niezbędne będą trzy kondensatory do gromadzenia ładunku).

Kondensator gromadzący ładunek ma typowo wartość 47 nF. Każda elektroda jest dołączona do mikrokontrolera przez rezystor zabezpieczający o typowej wartości 10 kΩ.

Do obsługi pól dotykowych służy blok TSC (Touch Sense Controller). Po skonfigurowaniu parametrów pracy i wybraniu jednego wejścia z każdej grupy pól oprogramowanie inicjuje pomiar pojemności. Dalsze czynności są wykonywane przez sprzęt – następuje seria cykli ładowania elektrod dotykowych i przelewania ładunku do kondensatorów. W każdym cyklu są inkrementowane liczniki cykli aktywnych grup. Proces ten jest powtarzany do osiągnięcia określonego napięcia na kondensatorze; licznik cykli grupy jest w tym momencie zatrzymywany. Proces pomiaru kończy się z chwilą osiągnięcia napięcia progowego na kondensatorach wszystkich aktywnych grup lub po osiągnięciu maksymalnej wartości liczników określonej przez oprogramowanie. Zakończenie pomiaru może powodować zgłoszenie przerwania. Moduł TSC rozróżnia dwie przyczyny przerwań – pomyślne zakończenie pomiaru i przekroczenie maksymalnej liczby cykli, którym odpowiadają dwa bity znaczników w rejestrze TSC–>ISR: EOA (End Of Acquisition) i MCE (Maximum Count Error).

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 31063, pass: 8iyw2174

Listing 2. Program demonstracyjny

```
/*
    STM32F0 tutorial
    F072BDISCO Simple Touch Sense
    gbm, 05'2014
*/
#include "stm32f0discovery.h"
// Timings
#define SYSCLK_FREQ  HSI_VALUE
#define BAUD_RATE    115200
// PWM constants
#define PWM_FREQ  400 // Hz
#define PWM_STEPS    80
#define   PWM_CLK    SYSCLK_FREQ
#define   PWM_PRE    (PWM_CLK / PWM_FREQ / PWM_STEPS)
#define LED_MAX    (PWM_STEPS - 1)
#define LED_DIM    1
#define LED_OFF    0
#define TSC_CR_CFGVAL      (TSC_CR_CTPHV(1) | TSC_CR_CTPLV(1) | TSC_CR_PGPSCV(3) | TSC_CR_MCVV(6) | TSC_CR_
TSCE)

void SystemInit(void)
{
}

static const struct init_entry_ init_table[] =
{
    // clock gating
    {&RCC->AHBENR, RCC_AHBENR_TSCEN | RCC_AHBENR_GPIOAEN | RCC_AHBENR_GPIOBEN | RCC_AHBENR_GPIOCEN | RCC_
AHBENR_RSTVAL},
    {&RCC->APB1ENR, RCC_APB1ENR_TIM3EN},    // TIM3
    // port setup
    {&GPIOA->AFR[0], BF4(7, 3) | BF4(6, 3) | BF4(3, 3) | BF4(2, 3)}, //
    {&GPIOA->MODER, GPIOA_MODER_SWD | BF2(7, GPIO_MODER_AF) | BF2(6, GPIO_MODER_AF) | BF2(3, GPIO_MODER_AF) |
BF2(2, GPIO_MODER_AF)},
    {&GPIOB->AFR[0], BF4(1, 3) | BF4(0, 3)},
    {&GPIOB->MODER, BF2(1, GPIO_MODER_AF) | BF2(0, GPIO_MODER_AF)},
    {&GPIOC->MODER, BF2(9, GPIO_MODER_AF) | BF2(8, GPIO_MODER_AF) | BF2(7, GPIO_MODER_AF) | BF2(6, GPIO_
MODER_AF)},    // set LED pins as AF
    // Touch sense
    {&TSC->IOSCR, TSC_CAPS},    // capacitors
    {&TSC->IOCCR, TSC_KEYS},    // inputs
    {&TSC->IOGCSR, 1 << 2 | 1 << 1 | 1 << 0},    // group enable
    {&TSC->IOHCR, ~(TSC_CAPS | TSC_KEYS)},   // hysteresis disable
    {&TSC->IER, TSC_IR_MCE | TSC_IR_EOA},    // enable ints
    {&TSC->CR, TSC_CR_CFGVAL | TSC_CR_START},   // config & start
    // PWM timer setup - TIM3
    {(__IO32p)&TIM3->PSC, PWM_PRE - 1},
    {(__IO32p)&TIM3->ARR, PWM_STEPS - 1},
    {(__IO32p)&LED_U_PWM, LED_DIM},
    {(__IO32p)&LED_D_PWM, LED_DIM},
    {(__IO32p)&LED_L_PWM, LED_DIM},
    {(__IO32p)&LED_R_PWM, LED_DIM},
    {(__IO32p)&TIM3->CCMR1, TIM_CCMR1_OC2M_PWM1 | TIM_CCMR1_OC2PE | TIM_CCMR1_OC1M_PWM1 | TIM_CCMR1_OC1PE},
// PWM mode 1, buffered preload
    {(__IO32p)&TIM3->CCMR2, TIM_CCMR2_OC4M_PWM1 | TIM_CCMR2_OC4PE | TIM_CCMR2_OC3M_PWM1 | TIM_CCMR2_OC3PE},
// PWM mode 1, buffered preload
    {(__IO32p)&TIM3->CCER, TIM_CCER_CC4E | TIM_CCER_CC3E | TIM_CCER_CC2E | TIM_CCER_CC1E},  // enable PWM
output
    {(__IO32p)&TIM3->DIER, TIM_DIER_UIE},  // enable update interrupt
    {(__IO32p)&TIM3->CR1, TIM_CR1_ARPE | TIM_CR1_CEN},  // auto reload, enable
    // interrupts and sleep
    {&NVIC->ISER[0], 1 << TIM3_IRQn | 1 << TSC_IRQn},  // enable interrupt
    {&SCB->SCR, SCB_SCR_SLEEPONEXIT_Msk},  // sleep while not in handler
    {0, 0}
};

int main(void)
{
    writeregs(init_table);
    __WFI();  // go to sleep
}

static uint8_t up_led_timer = 0;  // red LED timer
// LED intensities
static struct leds_ {
    uint8_t up, down, left, right;
} target = {LED_DIM, LED_DIM, LED_DIM, LED_DIM};

void TIM3_IRQHandler(void)
{
    static uint8_t tdiv;
    uint32_t pwmval;
    TIM3->SR = ~TIM_SR_UIF;  // clear interrupt flag
    if ((++ tdiv & 3) == 0)
    {
    // 100 Hz
        if (up_led_timer && -- up_led_timer == 0) target.up = LED_DIM;  // touch ack off
    }
    // change LED intensities
    if ((pwmval = LED_U_PWM) != target.up) LED_U_PWM = pwmval < target.up ? pwmval + 4 : pwmval - 1;
    if ((pwmval = LED_D_PWM) != target.down) LED_D_PWM = pwmval < target.down ? pwmval + 4 : pwmval - 1;
    if ((pwmval = LED_L_PWM) != target.left) LED_L_PWM = pwmval < target.left ? pwmval + 4 : pwmval - 1;
    if ((pwmval = LED_R_PWM) != target.right) LED_R_PWM = pwmval < target.right ? pwmval + 4 : pwmval - 1;
}

// Touch sense
#define ASHIFT    5  // low pass filter order
#define NTSKEYS   3
struct tsdata_ {
    uint32_t avg;
    uint8_t state;
```

Listing 2. c.d.

```
    _Bool det;
};
static struct tsdata_ tsdata[NTSKEYS];

static void process_touch(uint32_t key, uint32_t group)
{
    uint32_t v;
    tsdata[key].avg -= tsdata[key].avg >> ASHIFT;
    v = TSC->IOGXCR[group];
    tsdata[key].state <<= 1;
    tsdata[key].state |= v < (tsdata[key].avg >> ASHIFT);
    if ((tsdata[key].state & 3) == 1)
    {
        tsdata[key].det = 1;    // set key flag
        up_led_timer = 10;
        target.up = LED_MAX;    // fire LED
    }
    tsdata[key].avg += v;
}

void TS_IRQHandler(void)
{
    TSC->ICR = TSC_IR_MCE | TSC_IR_EOA;
    process_touch(0, 0);
    process_touch(1, 1);
    process_touch(2, 2);
    if (tsdata[0].det)
    {
        target.left ^= LED_MAX | LED_DIM;
        tsdata[0].det = 0;
    }
    if (tsdata[1].det)
    {
        target.down ^= LED_MAX | LED_DIM;
        tsdata[1].det = 0;
    }
    if (tsdata[2].det)
    {
        target.right ^= LED_MAX | LED_DIM;
        tsdata[2].det = 0;
    }
    TSC->CR = TSC_CR_CFGVAL | TSC_CR_START;    // start sampling
}
```

Po zakończeniu akwizycji oprogramowanie może odczytać wartości liczników cykli, dostępnych w postaci wektora TSC–>IOGXCR[].

Określanie stanu pola dotykowego

Sama wartość odczytana z rejestru IOGXCR nie niesie bezpośredniej informacji o tym, czy pole zostało dotknięte. Istnieje wiele możliwości konstruowania pól dotykowych i ich zespołów – mogą one zastępować niezależne przyciski albo suwaki lub pokrętła. W tym drugim, bardziej złożonym przypadku oprogramowanie musi określić, o ile bieżąca pojemność poszczególnych pól różni się od ich pojemności w stanie nieaktywnym i na tej podstawie wyliczyć pozycję palca. Firma ST Microelectronics oferuje gotową, rozbudowaną bibliotekę programową do obsługi różnego rodzaju interfejsów dotykowych.

W naszym przykładzie zajmiemy się pierwszym, prostszym przypadkiem – niezależnymi przyciskami. Wykrycie dotknięcia polega w tym przypadku na rozpoznaniu skoku pojemności pola dotykowego. Wartość pojemności elektrycznej pola reprezentowana przez licznik cykli transferu ładunku zależy od wielu czynników, w tym wykonania egzemplarza urządzenia i wilgotności otoczenia. Pojemności pól mogą zmieniać się podczas pracy urządzenia. Wiarygodne wykrycie dotknięcia nie może więc opierać się na prostym porównaniu wartości licznika cykli ze znaną z góry wartością. Podczas pracy urządzenie musi ciągle kalibrować odczyty. Można zrobić to na wiele sposobów. Technika zastosowana w bibliotece ST jest złożona i kosztowna obliczeniowo. Celem prezentowanego projektu było przedstawienie prostej obsługi interfejsu dotykowego, nieangażującej znacząco czasu procesora i możliwej do realizacji w przerwaniu.

W oprogramowaniu zastosowano prostą technikę filtra dolnoprzepustowego. Uśredniona wartość wyniku pomiaru jest aktualizowana w każdym cyklu akwizycji. Naciśnięcie pola jest wykrywane, gdy wynik pomiaru odbiega znacząco od wartości uśrednionej.

Przykład dla płytki STM32F072BDISCOVERY

Przykładowy projekt oprogramowania został zrealizowany na płytce 32F072BDISCOVERY. Dostępne na płytce pola dotykowe są obsługiwane w programie jako trzy niezależne przyciski. Ponieważ pola na płytce zostały zaprojektowane do symulacji suwaka, zachodzą on na siebie, co niekiedy skutkuje równoczesnym wykryciem naciśnięcia dwóch sąsiednich pól. Dwie skrajne strefy suwaka stanowią jedną, wspólną elektrodę dotykową. Na płytce Discovery każde pole należy do innej grupy, więc jest możliwy równoczesny pomiar ich pojemności.

Projekt oprogramowania, przygotowany w środowisku Keil MDK-ARM, nosi nazwę Touch i jest dostępny w aktualnej wersji zestawu programów przykładowych, dostępnych na serwerze EP w postaci pliku archiwum F0tutorial-xxx.zip.

Program realizuje funkcję trzech wyłączników bistabilnych, sterujących świeceniem trzech spośród czterech diod LED dostępnych na płytce. Kolejne naciśnięcia poszczególnych pól powodują naprzemienne włączanie i wyłączanie odpowiadających im diod. Czwarta, czerwona dioda służy do sygnalizacji wykrycia naciśnięcia któregokolwiek z pól. Wszystkie diody są sterowane przebiegami PWM generowanymi przez timer TIM3. Przerwanie timera służy do płynnego rozjaśniania i ściemniania diod. Mikrokontroler pracuje z zegarem wewnętrznym o częstotliwości 8 MHz. Obsługa pól dotykowych jest

realizowana całkowicie w przerwaniu TSC. Program nie zawiera pętli zdarzeń.

Plik stm32F0discovery.h został rozszerzony o definicje potrzebne dla płytki STM32F072BDISCO, w tym o przypisania diod LED do kanałów timera TIM3 oraz o wartości rejestrów TSC służące do konfiguracji linii interfejsu dotykowego.

Inicjowanie mikrokontrolera

Pierwszą czynnością w ramach sekwencji inicjującej jest włączenie używanych peryferiali – portów GPIO i modułu TSC oraz timera TIM3. Następnie następuje przyporządkowanie linii portów do peryferiali – linie sterujące diodami LED zostają skojarzone z wyjściami PWM timera, a linie pól dotykowych i kondensatorów gromadzących ładunek – z wyprowadzeniami TSC. Kolejnym etapem jest zainicjowanie sterownika TSC. Przy programowaniu rejestrów konfiguracyjnych TSC posługujemy się oznaczeniami linii TSC, mającymi postać Gx_IOy, gdzie x jest numerem grupy (1..8), a y – numerem linii w grupie (1..4). W celu uruchomienia modułu TSC należy kolejno:

- Włączyć moduł interfejsu dotykowego ustawiając bit TSCEN w rejestrze AHBENR.
- Ustawić w rejestrach AFR portów GPIOA i GPIOB funkcję interfejsu dotykowego (AF3) dla linii PA2, PA3, PA6, PA7, PB0 i PB1.
- Ustawić w rejestrach MODER funkcję AF dla wymienionych linii.
- W rejestrze TSC–>IOSCR wybrać linie, do których są podłączone kondensatory gromadzące ładunek – G3_IO3, G2_IO4, G1_IO4.
- W rejestrze TSC–>IOCCR wybrać linie odpowiadające polom dotykowym – G3_IO2, G2_IO3, G1_IO3.
- W rejestrze TSC–>IOGCSR wybrać aktywne grupy – G3, G2, G1.
- Wyłączyć histerezę wejść odpowiadających liniom używanym przez TSC – rejestr TSC>IOHCR.
- Włączyć możliwość zgłaszania przerwań od zakończenia zliczania i przekroczenia zakresu – rejestr TSC–>IER.
- Ustawić parametry akwizycji i uaktywnić moduł TSC poprzez zapis do rejestru TSC–>CR.

Parametry akwizycji zostały dobrane następująco:

- podział częstotliwości zegara – 8 (okres zegara TSC równy 1 µs),
- czasy ładowania i transferu ładunku – po 2 µs,
- maksymalna wartość licznika cykli: 16383.

Przerwania z TSC będą zgłaszane po zakończeniu każdej sekwencji akwizycji, nie rzadziej niż co 64 ms (w praktyce – kilkukrotnie częściej).

Po zaprogramowaniu TSC następuje zaprogramowanie timera TIM4, który będzie zgłaszał przerwania z częstotliwością odpowiadającą częstotliwości PWM – 400 Hz.

Na końcu sekwencji inicjującej następuje włączenie przerwań w module NVIC oraz włączenie usypiania procesora po wyjściu z obsługi przerwania. Po zainicjowaniu urządzenia wykonywane będą tylko dwie procedury obsługi przerwań.

Detekcja stanu pola

Do analizy stanu pola dotykowego i detekcji jego naciśnięcia służy procedura *process_touch()* wywoływana z procedury obsługi przerwania TSC. Procedurę napisano w sposób umożliwiający wielofazową akwizycję stanu dowolnej liczby pól w każdej grupie. W naszym przykładzie mamy do czynienia z akwizycją jednofazową – po jednym polu z każdej grupy.

Z każdym polem dotykowym jest związana struktura danych zawierająca:

- przefiltrowaną sumę pomiarów *avg*, z której przez przesunięcie bitowe można uzyskać uśrednioną wartość pomiaru.
- rejestr ostatnich ośmiu stanów pola dotykowego *state*,
- Znacznik wykrycia naciśnięcia pola *det*.

Procedura pobiera wartość licznika cykli transferu, porównuje ją z wartością uśrednioną i aktualizuje wartość uśrednioną. Ponieważ wykrycie naciśnięcia pola wymaga widocznego skoku wartości, w celu zmniejszenia liczby wykonywanych operacji zmieniono ich kolejność, eliminując potrzebę obliczania progu wykrycia naciśnięcia – najpierw wartość uśredniona jest pomniejszana o stały jej ułamek (co jest częścią obliczeń potrzebnych przy filtrowaniu), następnie jest wykonywane porównanie nowego odczytu z tak zmniejszoną wartością średniej, po czym średnia jest aktualizowana przez dodanie nowego odczytu. Wynik porównania w postaci jednego bitu jest wsuwany do rejestru *state*. Wykrycie na podstawie zawartości tego rejestru zmiany stanu pola powoduje ustawienie znacznika *det* i zaświecenie czerwonej diody LED, która zostanie wyłączona po upływie określonego czasu.

Obsługa przerwania TSC

Obsługa przerwania TSC rozpoczyna się od wyzerowania znaczników przerwań. Następnie dla każdego z obsługiwanych pól dotykowych jest wywoływana procedura analizy stanu pola *process_touch()*. Sekwencja instrukcji *if()* bada wartości znaczników naciśnięcia poszczególnych pól, a w przypadku wykrycia naciśnięcia zmienia stany odpowiadających im diod i zeruje znaczniki. Na końcu obsługi przerwania następuje zainicjowanie kolejnego cyklu akwizycji stanu pól.

Obsługa przerwania timera TIM3

Procedura obsługi przerwania timera służy do gaszenia diody sygnalizującej moment naciśnięcia przycisku oraz do sterowania płynną zmianą jasności wszystkich diod. Sterowanie jasnością następuje na podstawie porównania wartości pól struktury target, zawierającej zadane wartości jasności, z bieżącymi wartościami jasności przechowywanymi w rejestrach *CCRx* timera.

Podsumowanie

Przedstawiony przykład ilustruje prostą i niewymagającą czasochłonnych obliczeń metodę obsługi przycisków dotykowych, możliwą do realizacji w postaci nieblokującej -wyłączenie przy użyciu przerwań. Biorąc pod uwagę prostotę oprogramowania, pola dotykowe mogą stanowić nie tylko atrakcyjną metodę interakcji z użytkownikiem, lecz także zastępować przyciski służące do włączania funkcji diagnostycznych urządzenia, umożliwiając dzięki rezygnacji z elementów mechanicznych (przycisków, zwor) obniżenie kosztów i wymiarów urządzenia.

Grzegorz Mazur