STM32 – nieblokująca obsługa panelu dotykowego z układem XPT2046

Popularne, niedrogie wyświetlacze LCD TFT są zwykle wyposażone w rezystancyjny panel dotykowy współpracujący z układem XPT2046 lub podobnym (ADS7843). Artykuł przedstawia prosty sposób obsługi programowej panelu, z nieblokującym odczytem stanu i pozycji.

Dotykowy panel rezystancyjny pod względem elektrycznym jest równoważny dwóm potencjometrom ze zwartymi suwakami. Odczyt pozycji nacisku wymaga określenia pozycji obu suwaków, do których nie ma bezpośredniego połączenia elektrycznego. Odbywa się to w dwóch fazach: najpierw podajemy napięcie na jeden potencjometr i odczytujemy napięcie z jednego z końców drugiego potencjometru, a następnie podajemy napięcie na drugi potencjometr i odczytujemy wartość napięcia z jednego z końców pierwszego potencjometru. Rolę potencjometrów w panelu rezystancyjnym pełnią dwie warstwy folii; naciśnięcie panelu powoduje zwarcie obu warstw, a punkt naciśnięcia odpowiada pozycji suwaków potencjometrów.

Układ XPT2046

Używany typowo do współpracy z panelami rezystancyjnymi układ XPT2046 umożliwia łatwe przeprowadzenie wymaganych pomiarów pod kontrolą mikrokontrolera poprzez podawanie i pomiar napięć elektrody panelu. Oprócz podstawowego zastosowania, układ XPT2046 może wykonywać dodatkowe pomiary – temperatury, napięcia zasilania oraz siły nacisku (która wpływa na pole powierzchni styku obu warstw folii).

Układ XPT2046 zawiera klucze napięciowe, multipleksery analogowe oraz 12-bitowy przetwornik analogowo cyfrowy. Jest on wyposażony w interfejs SPI. Krawędzie dwóch warstw folii tworzących panel dotykowy dołącza się do przeznaczonych do tego czterech wejść/wyjść układu. Dodatkowo układ XPT2046 ma wyprowadzone wyjście sygnalizacji naciśnięcia panelu -PENIRQ, aktywne poziomem niskim.

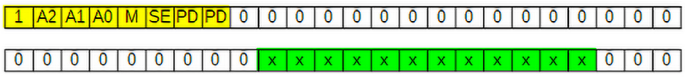
Przetwornik analogowo-cyfrowy XPT2046 jest taktowany zegarem interfejsu SPI. Logika układu została zaprojektowana w taki sposób, że użycie linii wyboru układu -CS jest niezbędne tylko wtedy, gdy z jednym interfejsem SPI mikrokontrolera współpracuje większa liczba układów. W przeciwnym przypadku linia -CS może być stale aktywna – układ reaguje na pierwszy niezerowy bit w ramce danych jako na pierwszy bit ramki, nie ma więc potrzeby sygnalizowania początku ramki przy użyciu linii -CS. Budowę ramek danych przesyłanych do i z układu XPT2046 pokazano na **rysunku 1**.

Ramka polecenia rozpoczyna się od bitu o wartości 1, po którym następują trzy bity sterujące pomiarem. W naszym zastosowaniu użyjemy tylko dwóch z ośmiu możliwych wartości, odpowiadających pomiarowi pozycji nacisku w osi X i Y. Kolejne 4 bity służą do wyboru trybu pracy układu; w prostych zastosowaniach mają one wartość 0.

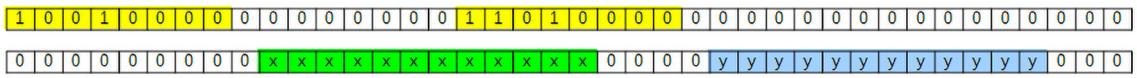
Podstawowy tryb akwizycji danych opisany w dokumentacji układu polega na transmisji ramek 24-bitowych. Ramka przesyłana do XPT2046 rozpoczyna się od 8 bitów polecenia, po których następuje 16 bitów o wartości 0. Ramka zwrotna rozpoczyna się od 9 zer, po których następuje 12 bitów wartości odczytanej

Listing 1. Procedura inicjowania akwiyzcji danych z panelu przy użyciu DMA

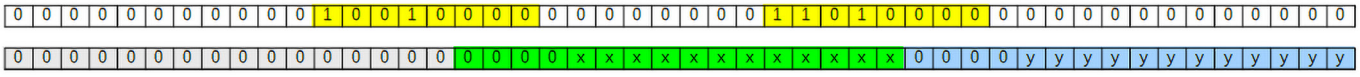
```
#define CHX  (0x90 >> 3)
#define CHY (0xd0 >> 3)

#define TP_TxDMACh    DMA1_Channel5
#define TP_RxDMACh    DMA1_Channel4
#define CLR_TPtx_FLAGS          (DMA1->IFCR = DMA_IFCR_CGIF4)

static uint16_t tp_rxbuf[3];

static void tp_hw_start_acquire(void)
{
    static const uint16_t cmd[3] = {CHY, CHX, 0};
    if (!TP_RxDMACh->CCR)
    {
        // rx - setup once, circular mode
        DMA1_CSELR->CSELR = BF4(4, 1) | BF4(3, 1);          // select SPI2
        TP_RxDMACh->CPAR = (uint32_t)&TP_SPI->DR;
        TP_RxDMACh->CMAR = (uint32_t)tp_rxbuf;
        TP_RxDMACh->CNDTR = 3;
        TP_RxDMACh->CCR = DMA_CCR_MSIZE16 | DMA_CCR_PSIZE16 | DMA_CCR_MINC | DMA_CCR_CIRC | DMA_CCR_EN;
    }
    // tx - DMA1 Channel 4 ch0
    TP_TxDMACh->CCR = 0;
    CLR_TPtx_FLAGS;
    TP_TxDMACh->CPAR = (uint32_t)&TP_SPI->DR;
    TP_TxDMACh->CMAR = (uint32_t)cmd;
    TP_TxDMACh->CNDTR = 3;
     TP_TxDMACh->CCR = DMA_CCR_MSIZE16 | DMA_CCR_PSIZE16 | DMA_CCR_MINC | DMA_CCR_DIR_M2P | DMA_CCR_EN;
}
```



Rysunek 1. Akwizycja jednego kanału w XPT2046 z transmisją 24 bitów



Rysunek 2. Akwizycja dwóch kanałów w XPT2046 z transmisją 40 bitów



Rysunek 3. Akwizycja dwóch kanałów w XPT2046 z transmisją trzech słów 16-bitowych

z przetwornika i 3 zera. Proces konwersji rozpoczyna się po przesłaniu piątego bitu i zajmuje 16 cykli zegarowych. Odczyt pozycji wymaga wykonania dwóch konwersji – dla współrzędnych X i Y, co można uzyskać przesyłając kolejno dwie ramki 24-bitowe.

Dokumentacja układu ukazuje również metodę odczytu serii wyników konwersji co 16 cykli zegara SPI. W takim trybie kolejne polecenia konwersji są przesyłane co 16 cykli zegara – polecenie rozpoczyna się jeszcze w czasie trwania poprzedniej konwersji (**rysunek 2**). Wadą obu rozwiązań pokazanych w dokumentacji układu jest konieczność wyłuskiwania wyników konwersji ze strumienia danych przy użyciu operacji przesunięć bitowych i maskowań. Należy przy tym zauważyć, że ponieważ dane są w interfejsie SPI transmitowane począwszy od najbardziej znaczącego bitu, składanie wyników 12-bitowych z ramek 8-bitowych wymaga w przypadku mikrokontrolerów używających konwencji adresowania little-endian dodatkowo zmiany kolejności bajtów.

Listing 2. Struktura danych i procedura odczytu stanu panelu wywoływana z przerwania timera

```
struct coords_
{
    int16_t y, x;
};

struct tpd_
{
    struct coords_ display;
    volatile _Bool touched, released;
};

struct tpd_ tp;

void tp_wait4touch(void)
{
    for (tp.touched = 0; !tp.touched;);
    tp.released = 1;
}

struct tp_mtx_
{
    int32_t An, Bn, Cn, Dn, En, Fn, Den;
};

static struct tp_mtx_ matrix;

#define TPDELTA 4
#define curr_tpad (*(struct coords_ *)(&tp_rxbuf[1]))

// tp sense, invoked from timer interrupt (50..100 Hz)
void tp_sense(void)
{
    static struct coords_ prev_tpad;
    static uint8_t tp_state = 0;
    static _Bool tp_acqd = 0;

    tp_state <<= 1;
    if (TP_NOT_TOUCHED)
    {
        // not touched
        if ((++ tp_state & 7) == 7)
        {
            tp.released = 1;
            prev_tpad.x = 0;
            prev_tpad.y = 0;
        }
        tp_acqd = 0;
    }
    else if (!tp.touched)
    {
        // was not touched, now touched
        static _Bool acquire = 0;
        acquire = 1;
        if (tp_acqd)
        {
            int16_t dif;
            if ((dif = curr_tpad.x - prev_tpad.x) < TPDELTA && dif > -TPDELTA
               && (dif = curr_tpad.y - prev_tpad.y) < TPDELTA && dif > -TPDELTA)
            {
                tp.display.x = (matrix.An * curr_tpad.x + matrix.Bn * curr_tpad.y + matrix.Cn) / matrix.Den;
                if (tp.display.x < 0) tp.display.x = 0;
                    else if (tp.display.x > DISP_MAX_X) tp.display.x = DISP_MAX_X;
                tp.display.y = (matrix.Dn * curr_tpad.x + matrix.En * curr_tpad.y + matrix.Fn) / matrix.Den;
                if (tp.display.y < 0) tp.display.y = 0;
                    else if (tp.display.y > DISP_MAX_Y) tp.display.y = DISP_MAX_Y;
                tp.touched = 1;
                acquire = 0;
            }
            prev_tpad = curr_tpad;
            tp_acqd = 0;
        }
        if (acquire)
        {
            tp_hw_start_acquire();
            tp_acqd = 1;
        }
    }
}
```

Bibliografia
1. XPZ2046 Touch Screen Controller Data Sheet, Shenzen XPTEK Technology Co., Ltd 2007
2. RM0351 Reference manual, STM32L4x5 and STM32L4x6 advanced ARM®-based 32-bit MCUs, ST Microelectronics 03'2017

Proponowany sposób współpracy z układem XPT2046

12-bitowe wyniki odczytane z XPT2046 jako liczby bez znaku powinny być reprezentowane w języku C przez dane typu int16_t (16-bitowe liczby całkowite ze znakiem), ze względu na to, że w późniejszym ich przetwarzaniu wyniki pośrednie mogą być liczbami ujemnymi. Konwersja dwóch kanałów z nakładaniem poleceń wymaga łącznie transmisji 40 bitów. W proponowanym rozwiązaniu do i z układu transmitujemy ramkę 48-bitową, złożoną z trzech słów 16-bitowych. Użyteczne dane odbierane są zawarte na 12 najmniej znaczących bitach

Listing 3. Procedury kalibracji panelu

```
// compute callibration matrix based on data sampled from 3 points
static _Bool SetCalMatrix(const struct coords_ *CalCoords, struct coords_ *TPsample, struct tp_mtx_ *MtxPtr)
{
    MtxPtr->Den = ((TPsample[0].x - TPsample[2].x) * (TPsample[1].y - TPsample[2].y))
        -((TPsample[1].x - TPsample[2].x) * (TPsample[0].y - TPsample[2].y)) ;
    if (MtxPtr->Den == 0) return 0;
    MtxPtr->An = ((CalCoords[0].x - CalCoords[2].x) * (TPsample[1].y - TPsample[2].y)) -
                 ((CalCoords[1].x - CalCoords[2].x) * (TPsample[0].y - TPsample[2].y)) ;
    MtxPtr->Bn = ((TPsample[0].x - TPsample[2].x) * (CalCoords[1].x - CalCoords[2].x)) -
                 ((CalCoords[0].x - CalCoords[2].x) * (TPsample[1].x - TPsample[2].x)) ;
    MtxPtr->Cn = (TPsample[2].x * CalCoords[1].x - TPsample[1].x * CalCoords[2].x) * TPsample[0].y +
                 (TPsample[0].x * CalCoords[2].x - TPsample[2].x * CalCoords[0].x) * TPsample[1].y +
                 (TPsample[1].x * CalCoords[0].x - TPsample[0].x * CalCoords[1].x) * TPsample[2].y ;
    MtxPtr->Dn = ((CalCoords[0].y - CalCoords[2].y) * (TPsample[1].y - TPsample[2].y)) -
                 ((CalCoords[1].y - CalCoords[2].y) * (TPsample[0].y - TPsample[2].y)) ;
    MtxPtr->En = ((TPsample[0].x - TPsample[2].x) * (CalCoords[1].y - CalCoords[2].y)) -
                 ((CalCoords[0].y - CalCoords[2].y) * (TPsample[1].x - TPsample[2].x)) ;
    MtxPtr->Fn = (TPsample[2].x * CalCoords[1].y - TPsample[1].x * CalCoords[2].y) * TPsample[0].y +
                 (TPsample[0].x * CalCoords[2].y - TPsample[2].x * CalCoords[0].y) * TPsample[1].y +
                 (TPsample[1].x * CalCoords[0].y - TPsample[0].x * CalCoords[1].y) * TPsample[2].y ;
          return 1;
}
// wait until panel touched
void tp_wait4touch(void)
{
    for (tp.touched = 0; !tp.touched;);
    tp.released = 1;
}
// draw a cross for sampling point
static void DrawCross(uint16_t Xpos,uint16_t Ypos)
{
    LCD_DrawLine(Xpos - 15, Ypos, Xpos - 2, Ypos, 0xffff);
    LCD_DrawLine(Xpos + 2, Ypos, Xpos + 15, Ypos, 0xffff);
    LCD_DrawLine(Xpos, Ypos - 15, Xpos, Ypos - 2, 0xffff);
    LCD_DrawLine(Xpos, Ypos + 2, Xpos, Ypos + 15, 0xffff);
}

#define CM_OFF 45
static const struct coords_ CalPoints[3] = {
    {CM_OFF, CM_OFF},
    {CM_OFF, DISP_MAX_X - CM_OFF},
    {DISP_MAX_Y - CM_OFF, DISP_MAX_X / 2}
};

void tp_calib(void)
{
    struct coords_ TPsamp[3];
    do
    {
        for (uint32_t i = 0; i < 3; i ++)
        {
            LCD_Clear(Black);
            LCD_Text(MAX_Y / 2, CH_W, "Touch crosshair to calibrate", 0xffff, Black);
            DrawCross(CalPoints[i].x, CalPoints[i].y);
            tp_wait4touch();
            TPsamp[i] = curr_tpad;
            for (tp.released = 0; !tp.released;);
        }
    }
    while (!SetCalMatrix(CalPoints, TPsamp, &matrix));
    LCD_Clear(Black);
}
```

drugiego i trzeciego odbieranego słowa, dzięki czemu są one poprawnie wyrównane i gotowe do użycia w obliczeniach. W celu uzyskania takiego wyrównanie danych polecenia konwersji muszą zaczynać 5 bitów przed odbieranym słowem danych, co można łatwo uzyskać przesuwając polecenia konwersji tak, by zaczynały się one od 12. i 28. bitu ramki 48-bitowej. Formaty ramek pokazano na **rysunku 3**.

Realizacja transmisji w STM32

Do akwizycji danych z układu XPT2046 można użyć interfejsu SPI współpracującego z modułem DMA. Maksymalna częstotliwość transmisji SPI wynosi 2 MHz. Interfejs jest inicjowany do transmisji danych 16-bitowych, co eliminuje konieczność programowej zmiany kolejności bajtów odbieranych z układu XPT2046.

Odczyt danych może być inicjowany przy końcu procedury obsługi przerwania timera, zgłaszanego z częstotliwością np. 100 Hz. Po wykryciu nacisku, sygnalizowanym przez układ XPT2046 przy użyciu linii -PENIRQ, procedura obsługi przerwania programuje kanał DMA w celu nadania trzech słów 16-bitowych zawierających polecenia konwersji. Inny kanał DMA jest programowany jednokrotnie na ciągły, powtarzany odbiór trzech słów do bufora cyklicznego. Ponieważ transmisja danych w SPI zachodzi równocześnie w obu kierunkach, odbiór będzie następował wyłącznie w czasie nadawania.

Procedurę inicjującą akwizycję w wersji dla mikrokontrolera STM32L476 pokazano na **listingu 1**. Przy kolejnym przerwaniu timera, o ile wyjście PENIRQ jest nadal aktywne, odczytane dane są interpretowane. Pierwszą fazą interpretacji danych jest porównanie wartości z dwóch kolejnych odczytów. Wartości odczytane uznaje się za wiarygodne tylko wtedy, gdy wartość bezwzględna różnicy pomiędzy poprzednim i bieżącym wynikiem nie przekracza określonego progu (np. 4); w przeciwnym przypadku dane są odrzucane. Procedurę obsługującą odczyty pokazano na **listingu 2**.

Kalibracja panelu

Ze względu na znaczne rozrzuty parametrów każdy panel rezystancyjny wymaga kalibracji. W wyniku kalibracji zostają wyznaczone współczynniki macierzy, służącej do przeliczenia odczytów przetwornika analogowo-cyfrowego na współrzędne ekranu wyrażone w pikselach. Zastosowanie macierzy uniezależnia przebieg obliczeń od sposobu podłączenia panela – macierz automatycznie uwzględnia ewentualną zamianę współrzędnych X i Y.

Po wyznaczeniu macierzy kalibracji powinna ona zostać zapisana w pamięci nieulotnej (np. w pamięci Flash mikrokontrolera). Podczas inicjowania oprogramowania następuje sprawdzenie ważności macierzy współczynników i, gdy jest ona nieważna, następuje kalibracja panelu.

Kalibracja polega na kolejnym wyświetleniu na ekranie trzech punktów i naciśnięciu przez użytkownika panelu np. cienkim pisakiem w każdym z wyświetlonym punktów. Punkty powinny być rozmieszczone daleko od siebie i blisko krawędzi wyświetlacza, jednak nie w samych jego rogach. Mogą to być np. dwa punkty w pobliżu rogów wyświetlacza leżących przy wspólnej dłuższej krawędzi i trzeci punkt położony w pobliżu środka przeciwległej dłuższej krawędzi. Procedura kalibracji została pokazana na **listingu 3**.

Po odczytaniu wartości odpowiadających takim trzem punktom i zapisaniu ich w wektorze xxx następuje wyznaczenie macierzy transformacji. W celu uniknięcia obliczeń zmiennopozycyjnych współczynniki macierzy są zapisywane w postaci liczników ułamków i ich wspólnego mianownika (list. 3). Podczas normalnej pracy urządzenia odczyty przetwornika układu XPT2046 są przeliczane na współrzędne ekranu z zastosowaniem wzorów:

$$x_s = (x_p \cdot A + y_p \cdot B + C) / den$$

$$y_s = (x_p \cdot D + y_p \cdot E + F) / den',$$

Zmienne x_s i y_s oznaczają współrzędne ekranu, a x_p i y_p – odczyty wartości przetwornika XPT2046. Fragment kodu realizujący to zadanie, stanowi część procedury akwizycji z list. 2. Istotnymi zaletami zaprezentowanego rozwiązania są brak oczekiwania programowego na odczyt danych oraz minimalizacja narzutu czasu na obliczenia współrzędnych. Dzięki temu przedstawione fragmenty kodu mogą być użyte w oprogramowaniu bazującym wyłącznie na procedurach obsługi zdarzeń (przerwań) i nie zawierającym pętli zdarzeń.

Grzegorz Mazur