Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska

Technika Mikroprocesorowa

Sprawozdanie z laboratorium nr 4
Miernik czasu reakcji, przycisk rozpoczyna i kończy pomiar, przy użyciu mikrokontrolera z rodziny MSP430

Jakub Sikora
Konrad Winnicki
Marcin Dolicher

Warszawa, 20 maja 2018

Spis treści

1. Zadanie laboratoryjne

1.1. Polecenie

Celem czwartego zadania laboratoryjnego było zaprojektowanie, złożenie, zaprogramowanie i przetestowanie układu z mikrokontrolerem MSP430F16x tak aby działał on jako miernik czasu reakcji – jeden przyciski rozpoczyna pomiar, drugi kończy pomiar. Zasada działania podobna jak w przypadku zwykłych stoperów. Wartość mierzonego czasu powinna być na bieżąco wyświetlana na wyświetlaczu dynamicznym, który obowiązkowo obsługiwany jest za pomocą przerwań (jedno przerwanie - jedno przełączenie pozycji).

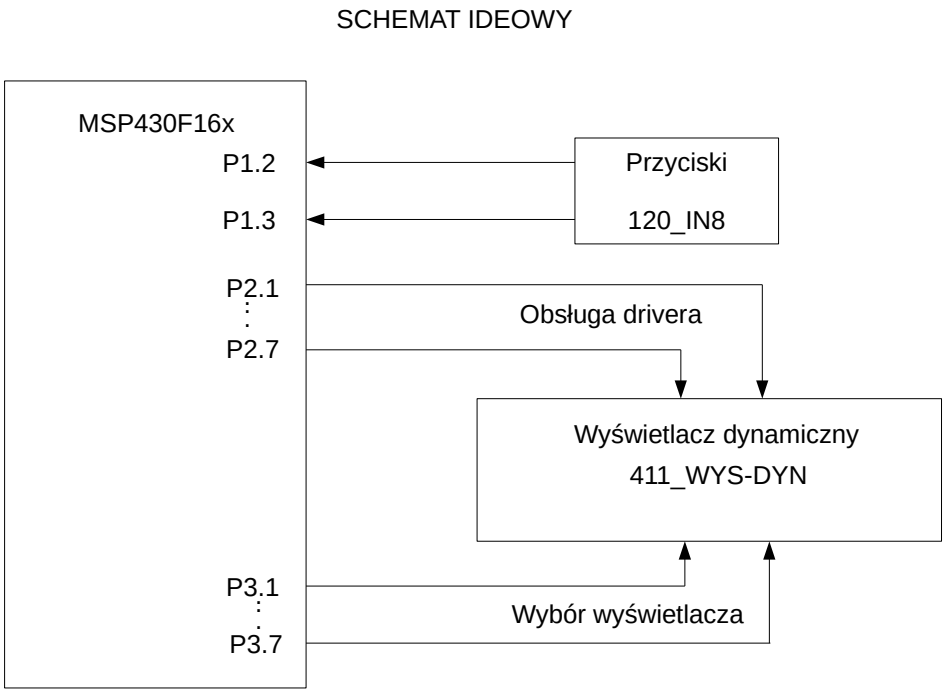
1.2. Szczegółowe uwagi

Częstotliwość odświeżania powinna gwarantować wyświetlenie każdej cyfry co najmniej 50 razy na sekundę. Do otrzymania satysfakcjonującej dokładności pomiaru długości trwania impulsów wymagane jest użycie trybu CAPTURE i obliczanie interwału z uwzględnieniem odczytów licznika przy zboczu początkowym i końcowym oraz liczby przerwań TxIFG, jakie zarejestrowano pomiędzy tymi zboczami. Kod programu został odpowiednio podzielony na część realizującą obsługę sprzętową i część aplikacyjną. Interakcję z systemem wzbogaciliśmy o wyświetlanie zer przy starcie systemu i przecinka, oddzielającego liczbę milisekund od sekund.

2. Projekt licznika

2.1. Opis sprzętu

Dzięki strukturze mikrokontrolera MSP43016x podłączenie urządzeń peryferyjnych nie było skomplikowanym zadaniem, jednak wymagało uwagi, aby nie pomylić portów obsługujących wyświetlacz.



Rysunek 2.1. Schemat ideowy

Do portu pierwszego podłączyliśmy monostabilne wyłączniki (120_IN8), które posłużyły nam jako źródło sygnałów wejściowych, odpowiednio dla przycisku start – P1.2, stop – P1.3. Dla wyświetlacza zostały zarezerwowane porty P2.0 – P2.7 (obsługa drivera) i P3.0 – P3.7 (wybór wyświetlacza 7-segmentowego). W Timer_A wykorzystujemy wszystkie trzy kanały, dwa w trybie capture i jeden w trybie compare.

2.2. Opis oprogramowania

Do zmniejszenia poboru energii użyliśmy trybu LPM0, ponieważ tylko on gwarantował nam brak zmian częstotliwości pracy zegara SMCLK. Kod programu napisaliśmy w języku C, co było niewątpliwym ułatwieniem w porównaniu do poprzednich laboratoriów. Pozwoliło to na

lepsze uporządkowanie funkcjonalności i zadań do wykonania w naszym programie. Kod stał się bardziej przejrzysty i zrozumiały.

2.2.1. Zliczanie czasu

Naciśnięcie przycisku może nastąpić w momencie gdy jesteśmy na zboczu zegara, dlatego do dokładnego obliczania zmierzonego czasu zastosowaliśmy przerwania w trybie capture, dzięki którym jesteśmy w stanie określić pozycję w jakiej byliśmy na tym zboczu. Zbocza liczników są opadające. Po uruchomieniu systemu i naciśnięciu przycisku – start, stoper zaczynał odliczanie. Zapamiętujemy miejsce na zboczu i zaczynamy zliczać ilość przepełnień licznika w celu obliczenia zmierzonego czasu. Przy naciśnięciu przycisku – stop, znowu zapamiętujemy miejsce na zboczu i używając wcześniej zapisanych danych obliczamy dokładny czas do wyświetlenia.

$$n_{cykli} = t_{konc} + (n_{overflow} + 1) * n_{graniczna} - t_{pocz}$$

2.2.2. Obsługa wyświetlacza

W celu taktowania odświeżania wyświetlacza dynamicznego zastosowaliśmy przerwania generowane w trybie `compare` z Timera A. Przerwanie generuje się co określony takt czasu, w momencie gdy wartość licznika TAR zrównuje się z wartością zapisaną w kanale TACCR0, to ustawiana jest flaga przerwania (bit CCIFG w rejestrze TACCTL0) Ustawiona częstotliwość odświeżania to 1kHz. Zasada działania wyświetlacza dynamicznego jest następująca: najpierw wybieramy wyświetlacz, który będzie podświetlony, wypisujemy wartość i podtrzymujemy ją do przyjścia kolejnego przerwania. Aby wyświetlacz sprawiał wrażenie stałego, każdy segment powinien być odświeżany z częstotliwością większą niż 50 Hz.

Zasadniczym problemem podczas odświeżania wyświetlacza jest poprawne aktualizowanie wartości. Mając na uwadze zastosowanie aplikacyjne naszego rozwiązania zdecydowaliśmy się na liczenie czasu na dwa sposoby. Podczas aktywnego zliczania długości trwania impulsu, wyświetlana jest wartość zgrubna, obliczana z pomocą przerwań z trybu `compare`. Gdy użytkownik wciśnie przycisk `STOP` to wyświetlana jest wartość dokładna obliczana z wzoru przedstawionego w sekcji powyżej.

Aby czas wykonania przerwania nie był za długi przy wpisywaniu dokładnej wartości do tablicy stosujemy dzielenie modulo przez 10, co zapewnia nam satysfakcjonujące efekty i pozwala na szybkie wykonanie przerwania. Dodatkowo wyświetlanie wartości aktualnej jest aktualizowane co drugie przerwania, żeby maksymalnie, oczywiście w granicach normy skrócić czas wykonania przerwania.

Podczas fazy testów, za pomocą oscyloskopu sprawdziliśmy ile trwają przerwania podczas aktywnego zliczania i podczas oczekiwania na start. W trakcie oczekiwania na start, przerwanie obsługuje tylko zmianę segmentu i wartości na wyświetlaczu. Przerwanie zajmuje wtedy około 20 % czasu. W przypadku gdy zliczanie jest aktywne to czas obsługi przerwania wydłuża się do około 30 % dostępnego czasu.

Rysunek 2.2. Oscylogram przedstawiający czas trwania przerwań bez aktywnego zliczania

Rysunek 2.3. Oscylogram przedstawiający czas trwania przerwań z aktywnego zliczania

2.2.3. Kod programu

Poniżej znajduje się kod programu napisanego w języku C:

```
#include <msp430.h>
#include <stdio.h>
#include <stdint.h>

#define bool uint_t
#define true 1
#define false 0

    // Bity sterujace driverem 7-SEG LED
#define LED_DP (1«7)
#define LED_LT (1«6)
#define LED_BI (1«5)
#define LED_RBI (1«4)

    // Ilosc wyswietlanych znakow
#define SEG_LED_NUMBER 7
#define SEG_LED_DOT_POSITION 5
#define SEG_LED_DISPLAY_SYSTEM 10

    // Watrość rejestru licznika
#define TACCR0_VALUE 720

volatile char buffer[SEG_LED_NUMBER]; // BUFOR ZLICZANIA CZASU REAKCJI
volatile uint32_t count_time = 0; // wartosc czasu reakcji
volatile bool if_counting = false;
volatile bool end_of_counting=false;

typedef struct captured_cycles
{
int start_time;
int num_of_overflows;
int end_time;
} captured_cycles;

int get_cycles(captured_cycles time)
{
return (time.num_of_overflows + 1)*TACCR0_VALUE + time.end_time - time.start_time;
}

int get_time(int cycles)
{
return (int)(cycles/10);
}

void temp_time()
{
uint16_t n = SEG_LED_DOT_POSITION-3; // zgrubny czas z dokładnością milisekundy
buffer[n]++; // odśwież pozycję milisekund
```

```
while( n¡SEG_LED_NUMBER )
{
if( buffer[n]¿=SEG_LED_DISPLAY_SYSTEM )
{
buffer[n]=0;
buffer[n+1]++;
n++;
}
else
break;
}
}

void copy_buffer(char disp_buffer[])
{
uint16_t n=0;
while( n¡SEG_LED_NUMBER )
{
disp_buffer[n] = buffer[n]; // buffer mamy globalny
n++;
}
}

void display(uint16_t disp, char disp_buffer[])
{
//static uint16_t disp = 0;
P3OUT =  (1«disp); //aktywujemy kolejny wyswietlacz
P2OUT = ((disp_buffer[disp])&0x0F) | LED_RBI | LED_BI | LED_LT | LED_DP; // wyswietlenie cyfry oraz bity sterujace
if(disp == SEG_LED_DOT_POSITION) P2OUT &=  LED_DP; // zapalenie kropki na odpowiedniej pozycji
//disp++; // wybor kolejnego wyswietlacza
//if(disp ¿= SEG_LED_NUMBER) disp = 0;
}

void prepare_to_display_ideal_value(captured_cycles cap_time)
{
int locked_time = get_time(get_cycles(cap_time)); //zatrzaskujemy wartosc na czas sekwencji wyswietlania

    uint_t n=0;
while(n¡SEG_LED_NUMBER)
{
buffer[n]=locked_time % 10;
locked_time /= 10;
n++;
}
}

volatile captured_cycles cap_time = {0, 0, 0};
```

```
int main(void)
{
// wylaczenie watchdoga
WDTCTL = WDTPW + WDTHOLD;

    // inicjalizacja portu P1
P1SEL = (BIT2 + BIT3); // ustaw P1.0 i P1.7 jako wejscia timera
P1DIR &=  (BIT2 + BIT3); // ustaw jako wejscia
// piny portu 1 służące do debugowania zależności czasowych przy pomocy oscyloskopu
P1DIR |= BIT7 | BIT6;
P1OUT |= BIT7 | BIT6;

    // inicjalizacja portu P2
P2SEL = 0x00; // ustaw caly port 2 jako GPIO
P2DIR = 0xFF; // ustaw port 2 jako wyjscia
P2OUT = (0x08&0x0F) | LED_RBI | LED_BI | LED_LT /*| LED_DP*/; // wyswietlenie ósemki
z kropką

    // inicjalizacja portu P3
P3SEL = 0x00; // ustaw caLy port 3 jako GPIO
P3DIR = 0xFF; // ustaw port 3 jako wyjscia
P3OUT = 0x00; // aktywuj wszystkie wyświetlacze

    // Inicjacja kanalu 0, compare
TACCR0 = TACCR0_VALUE; // 1 kHz
TACCTL0 = CCIE; // CCR0 interrupt enabled -OK -zalaczamy przerwania do timerow (jak
nie zadziala wlaczac osobna TA i TB)
// Inicjacja kanalu 1, capture, przycisk start
TACCTL1 = CM_2 + SCS + CAP + CCIE + CCIS_0; // falling edge, synchronus capture,
CCI1A
// Inicjacja kanalu 2, capture, przycisk stop
TACCTL2 = CM_0 + SCS + CAP + CCIE + CCIS_0; // falling edge, synchronus capture,
CCI2A
// Inicjacja Timera A
TACTL = TASSEL_2 + MC_1 + ID_0 + TAIE; // SMCLK/8, upmode -OK SMCLK = 1MHz;
MC_1 - UP MODE; ID_0 - dzielnik /1 (nie musi byc, ale niech bedzie)

    _bis_SR_register(CPUOFF+GIE); // LPM0, globalne wlaczenie obslugi przerwan

    while(true)
{

    }

    return 0;
}

#pragma vector=TIMERA0_VECTOR // TIMERA0_VECTOR - wektor do obslugi compare,
kanal 0
__interrupt void timerA0_ISR(void) // timer 1kHz
{
```

```
cap_time.num_of_overflows++;
static uint16_t disp = 0; //wybór aktywnego wyświetlacza
static uint16_t led_timer = 0; //timer programowy wyświetlacza dynamicznego
static char disp_buffer[SEG_LED_NUMBER]; // Bufor wyswietlanych znakow

    if(end_of_counting==true )
{
prepare_to_display_ideal_value(cap_time);
//display(disp, disp_buffer);
end_of_counting = false;
}

    if( if_counting==true )
{
temp_time();

    if(disp==0)
{
copy_buffer(disp_buffer);
}
}

    if(led_timer==0) // start sekwencji wyswietlania, 1kHz/2
{
led_timer=1;
display(disp, disp_buffer);
disp++;
if(disp¿= SEG_LED_NUMBER) disp = 0
; }else
led_timer–;

    //P1OUT &=  0x80;
}

#pragma vector=TIMERA1_VECTOR // TIMERA1_VECTOR - wektor do obslugi capture,
kanal 1, 2
__interrupt void timerA1_ISR(void)
{
switch(TAIV) //odczyt TAIV
{
case 2: //flaga CCIFG
P1OUT |= 0x40;
if_counting = true;
cap_time.start_time = TACCR1;
cap_time.num_of_overflows = 0;
cap_time.end_time = 0;

    TACCTL1 &=  CM_2;
TACCTL2 &=  CM_0;
TACCTL1 |= CM_0;
```

```
TACCTL2 |= CM_2;

    //end_of_counting=false;

    // zerownie buffer
uint16_t n=0;
while(n¡SEG_LED_DOT_POSITION)
{
buffer[n] = 0;
n++
}

    break; //zrodlo TACCR1

    case 4: //flaga CCIFG
P1OUT |= 0x40;
if_counting = false;
cap_time.end_time = TACCR2;

    TACCTL1 &=  CM_0;
TACCTL2 &=  CM_2;
TACCTL1 | = CM_2;
TACCTL2 |= CM_0;

    end_of_counting=true;
break; //zrodlo TACCR2
}
}
```