Zaawansowane możliwości układów FTDI (1)

Obsługa pamięci MTP za pomocą mikrokontrolera, komunikacja SPI/I²C

Mimo wyposażenia wielu mikrokontrolerów w wbudowane interfejsy USB, układy scalone konwerterów USB/UART są nadal często używane. Niestety, taki konwerter zajmuje jeden interfejs UART. Ponadto, do szybkiej komunikacji (ponad 115200 bps) mikrokontroler musi być taktowany kwarcem „uartowym". Co prawda, są oferowane również konwertery z interfejsem równoległym (FT240, FT245), ale wymagają użycia dużej liczby wyprowadzeń mikrokontrolera. Rozwiązaniem są mostki USB/I²C lub SPI mające dużo większe możliwości, niż te z UART.

Poza popularnymi mostkami USB-UART (FT232 i jego odmianach, FT230) oraz SPI-Pararrel (FT245, FT240) USB-SPI/I²C w ofercie dostępne są inne układy. Pomijając konwertery FT260 obsługujące klasę HID (klawiatura, mysz) dostępne są dwie rodziny: FT12x i FT22x/20x.

Układy **FT12x** wymagają konfiguracji, odczytania identyfikatora nadanego przez system (enumeracja), odpowiedzi swoim numerem VID, PID, zapotrzebowaniem na prąd. Układy dają duże możliwości – można otworzyć kilka kanałów logicznych (jak portów w Ethernecie), realizować szybkie transmisje o wysokim priorytecie. Niestety, są kłopotliwe w konfiguracji, dlatego na razie nie będziemy się nimi zajmowali.

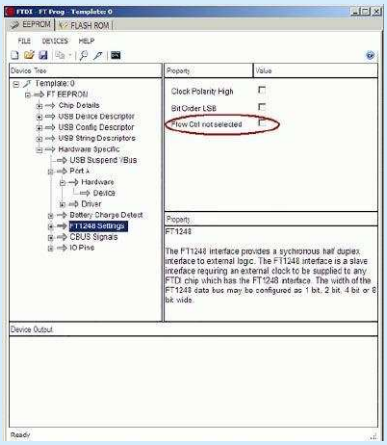
Układy **FT22x/20x** są łatwe w implementacji. Poza tym, że zamiast przez UART czy interfejs równoległy komunikują się z użyciem SPI lub I²C, umożliwiają dostęp do pamięci konfiguracji (MTP). Dzięki temu, że tę pamięć można zarówno czytać, jak i zapisywać, to z poziomu aplikacji mikrokontrolera jest możliwość zmiany konfiguracji układu (VID, PID, nazwa producenta, numer seryjny, funkcje linii CBUS). Ponadto, aplikacja ma dostęp do wolnej przestrzeni pamięci EEPROM (prawie 1 kB!). W tej pamięci można przechowywać konfigurację programu mikrokontrolera (np. backup wewnętrznej EEPROM), licznik czasu pracy programu i inne dane. Zaletą takiej pamięci jest, że nie jest kasowana po wgraniu programu do mikrokontrolera. Ponadto, za pomocą bibliotek D2XX można z poziomu systemu zapisywać i odczytywać pamięć MTP.

Układy FT220 i 221

Komunikacja za pomocą SPI w trybie 1-bit, 2-bit, 4-bit (jak w wypadku kart SD) oraz 8-bit (FT221). Od strony USB 1 Mb/s (FT221) i 500 kb/s (FT220). Komunikacja z układem jest banalnie łatwa. Najpierw jest przesyłana komenda, a po niej zapisywane/odczytywane dane. W wypadku odbierania danych z komputera, po wysłaniu komendy należy sprawdzić stan linii MISO układu (nie mylić z linią MIOSIO0), co pokazano na **listingu 1**. MISO informuje czy w buforze jest choć jeden znak do odczytu. Można odczytać wiele bajtów, jeśli oczywiście są do odebrania (**listing 2**).

Listing 1. Pobranie znaku z układu FT22x

```
word GetFT22x()
{
    byte znak, st;
    SPI_SSl();  // SS = L
    FT22xout( cmdFT22x_GetChar );  // Read request command (CMD=0x01 0x0001)
    if ((st=RD_FT22x()))  // Odczytaj stan MISO
    {  // Nie ma znaku
        SPI_SSh();  // SS = H
        return( NONE );  // Wyjście z -1 (word = unsigned int)
    }
    znak = FT22xin();  // Odczyt linii MIOSIO
    SPI_SSh();  // SS = H
    return( znak );
}
```



Rysunek 1. Wyłączenie funkcjonalności linii MIOSIO0 w roli statusu zapełnienia bufora nadawczego za pomocą FTProg

Przy zapisie sytuacja jest nieco inna. Linia MIOSIO0 (nie mylić z linią MISO) informuje czy bufor nadawczy jest pełny. Trzeba pamiętać, że stan tej linii jest ważny przed uaktywnieniem układu sygnałem strobującym CS (SS). W większości mikrokontrolerów zapełnienie bufora nadawczego jest w mało prawdopodobne i można ignorować stan tej linii. Istotne jest to, że po wysłaniu komendy można wysłać dowolną (do zapełnienia bufora) liczbę znaków. Dlatego, jeśli wysyłamy kilka bajtów, można to zrobić w sposób pokazany na **listingu 3**.

Taki system komunikacji jest dobry gdy na magistrali SPI jest jeden układ. Spowodowane jest to tym, że linia MIOSIO0 może zakłócać transmisję innych układów. Aby tego uniknąć a nie rozbudowywać sprzętu (bramki truj-stanowe) funkcjonalność linii MIOSIO0 w roli statusu zapełnienia bufora nadawczego, można wyłączyć z poziomu aplikacji FT_PROG (**rysunek 1**).

Układy FT22x umożliwiają zapis i odczyt linii statusowych modemu (RTC, CTS, DTR, DSR, CDC, RI). Jest to o tyle przydatne, że w Windows (sprawdziłem w XP, problem sygnalizowano mi także w Windows 7) ma pewien błąd. Gdy mostek FTDI wysyła paczki po kilkanaście bajtów do hosta, a nie jest otworzony port (wirtualny COM lub biblioteką D2XX), np. otworzony program terminala, to Windows może traktować takie dane jako informacje z klawiatury/myszki. Powoduje to, że kursor zaczyna poruszać się w przypadkowy sposób, otwierają się okna, bufor klawiatury

Listing 2. Odczyt znaków z układu FT22x
```
FT22xout( cmdFT22x_GetChar );  // Read request command (CMD=0x01 0x0001)
while (!(st=RD_FT22x()))  // Odczytaj stan MISO
{  // Nie ma znaku
    *buf++ = FT22xin();  // Odczyt linii MIOSIO
}
SPI_SSh();  // SS = H
```

Listing 3. Funkcja przesyłająca ciąg znaków
```
byte PrintStringFT22x( byte *text )
{
char znak, st;
    SPI_SSl();
    FT22xout(cmdFT22x_WrChar);  // Write request command (CMD=0x00)
    st = RD_FT22x();
    if ( st ){ SPI_SSh(); return( false ); }
    while (*text)  // Wyswietl tekst z RAMu
    {
        znak = (*text);
        FT22xout( znak );
        text++;
    }
    SPI_SSh();
    return( true );
}
```

Listing 4. Wysłanie bajta do FT201
```
byte PutCharFT201( byte znak )
{
    byte st;
    //----- START -----//
    st = TWI_Start();
    if (st != 8 && st != 0x10) return(false);
    //str.27        „AN_255_USB to I2C Example using the FT232H and FT201X devices”
    st = TWI_Write(FT201WR);
    if (st != 0x18)
    {
        TWI_Exit();
        return(false);
    }  // Brak ACK to bufor nadawczy pełny
    st = TWI_Write(znak );
    if (st != 0x28)
    {
        TWI_Exit();
        return(false);
    }
    //----- STOP -----//
    st = TWI_Stop();
    return(true);
}
```

Listing 5. Odczytanie znaku z bufora FT201
```
word ReadDataFT201()
{
    byte st, dana=NONE;
    //----- START -----//
    st = TWI_Start();
    if (st != 8 && st != 0x10) return(NONE);
    st = TWI_Write( FT201RD );
    // if ( st == 0x48 ){ TWI_Exit(); return( NONE ); }     // Brak ACK oznacza brak znaku do odbioru
    if (st != 0x40)
    {
        TWI_Exit();
        return(NONE);
    }
    // Odczytaj ostatni bajt
    dana = TWI_Read( false );
    if (TWI_Status != 0x58)
    {
        TWI_Exit();
        return( NONE );
    }
    //----- STOP -----//
    st = TWI_Stop();
    return(dana);
}
```

przepełnia się. W Linuksie nie stwierdziłem takich problemów. Błąd w Windows można łatwo naprawić. Każde otwarcie portu powoduje uaktywnieni linii DTR i RTS bez względu na to czy sterowanie przepływem jest włączone, czy nie. W układach FT22x można programowo odczytać stan tych linii i nie wysyłać danych, jeśli port nie jest otwarty. Warto też wspomnieć o możliwości generowania przerwań w razie pojawienia się znaku w odbiorniku linią CBUS.

Układy FT201, FT200

Układy mają interfejs I²C przesyłający dane z prędkością 3,4 Mb/s (FT200) lub 400 kb/s (FT201). Interfejs USB pracuje z prędkością 1 Mb/s, realny transfer to 1 Mb/s w FT200 i 200 kb/w w FT201. Mniejsza prędkość transmisji FT201 wynika z faktu, że przy przesłaniu

Listing 6. Odczyt liczby znaków w buforze FIFO
```
byte AvailableFT201()
{
    byte st, ava, repeat=FT201REPEAT;
    do
    {
        //----- START -----//
        st = TWI_Start();
        if (st != 8 && st != 0x10) continue;
        // General adres
        st = TWI_Write(0);
        if (st != 0x18)
        {
            TWI_Exit();
            continue;
        }
        //str.25 „AN_255_USB to I2C Example using the FT232H and FT201X devices”
        //DataAvailable command
        st = TWI_Write(0x0C);
        if (st != 0x28)
        {
            TWI_Exit();
            continue;
        }
        //----- START -----//
        st = TWI_Start();
        if (st != 8 && st != 0x10) return(0);
        st = TWI_Write(FT201RD);
        if (st != 0x40)
        {
            TWI_Exit();
            continue;
        }
        // Odczytaj ostatni bajt
        ava = TWI_Read(false);
        if (TWI_Status != 0x58)
        {
            TWI_Exit();
            continue;
        }
        //----- STOP -----//
        st = TWI_Stop();
        return(ava);
    } while(repeat--);
    return 0;
}
```

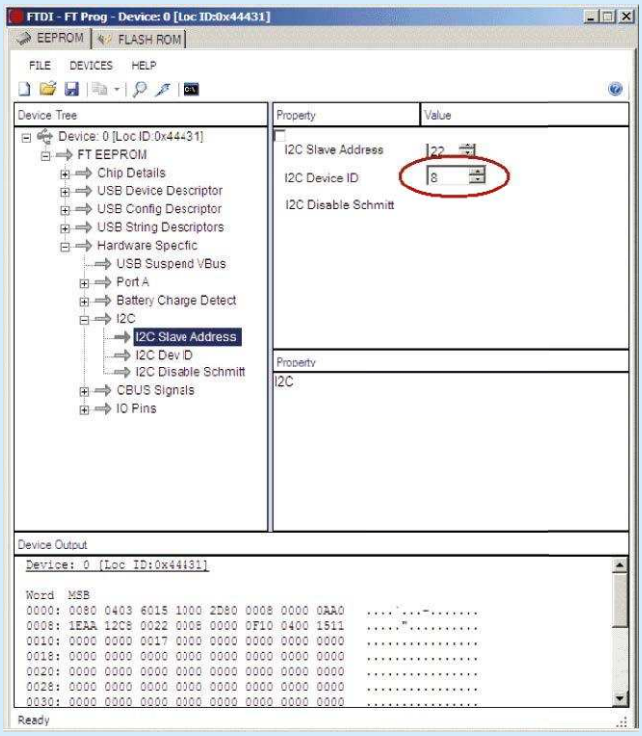
każdej danej układ należy zaadresować i nie można (jak w przypadku układów z interfejsem SPI) wysłać komendy i dowolnej liczby danych. Nadawanie/odbiór z USB jest banalnie łatwe. Przy nadawaniu, po zaadresowaniu układu (domyślny adres 22) sprawdzamy czy układ potwierdził obecność ACK. Jeśli tak, to bufor nadawczy może odebrać najmniej jeden znak (**listing 4**). Przy odczycie, jeśli po zaadresowaniu układu do odczytu otrzymamy ACK oznacza to, że w buforze odbiorczym FIFO znajduje się co najmniej jeden znak do odczytu (listing 5).

Czyli na dobrą sprawę nie ma jak sprawdzić czy układ jest obecny na magistrali. Sygnał ACK przy odczycie jest tylko, gdy w FIFO znajduje się znak. Przy zapisie nie ma gwarancji, że układ skomunikował się z hostem (wtedy też nie da ACK), a jeśli tak, to znak zostanie wysłany przez USB co czasem może być niekorzystne (problem zamkniętego portu w Windows). Konstruktorzy układu, aby nie zmniejszać przepustowości komunikacji przez dodatkowe wysłanie komendy wykorzystali adres rozgłoszeniowy (broadcast). Co ciekawe, do niektórych operacji jest używany adres 0x7C.

Układ w sumie zajmuje aż 3 adresy I²C. Broadcast to adres 0x00. Pod ten adres jest wysyłana komenda dla FT20x. Komenda jest potwierdzona przez wszystkie układy slave obsługujące broadcast. Następnie pod adres układu (domyślnie 22) wysyłane/odbierane są dane. Przykład odczytu liczby znaków w FIFO pokazano na **listingu 6**.

Na **listingu 7** pokazano przydatną funkcję, za której pomocą można sprawdzić status układu. Taka funkcja istnieje także dla układu FT22x. Dzięki niej można dowiedzieć się czy można już wysyłać odbierać dane przez USB (czy układ jest skonfigurowany, czy zakończono proces enumeracji).

Do identyfikacji układu, poza deskryptorem, można użyć 3 bajtów ID, przy czym jeden można ustawić z poziomu FT_PROG (**rysunek 2**). Mikrokontroler może odczytać ID używają funkcji z **listingu 8**. Wyróżniony fragment „0xF8/*0x7C*/” jest pewną niekonsekwencją w sposobie komunikacji z układem. W tym wypadku



Rysunek 2. Ustawienie bajtu deskryptora za pomocą FTProg

Memory Area Description	Word Address	Byte Address
User Area 2 Accessible via USB and I²C	0x3FF - 0x80	0x7FF – 0x100
Checksum	0x7F	0xFF – 0xFE
String Descriptor Area Accessible via USB and I²C	0x7D - 0x50	0xFB – 0xA0
FTDI Configuration Area Cannot be written	0x4F - 0x40	0x9F – 0x80
User Area 1 Accessible via USB and I²C	0x3F - 0x12	0x7F – 0x24
Chip Configuration Area Accessible via USB and I²C	0x11 - 0x00	0x23 - 0x00

Figure 3.1: Simplified memory map for the FT-X

Rysunek 3. Podział pamięci MTP

adres układu jest na stałe ustawiony przez producenta na 0x7C. Czemu tak to zrobiono? Nie jestem w stanie zrozumieć.

Pamięć MTP

Dostęp do tej pamięci zapewniają zarówno konwertery IIC jak i SPI. Dla ułatwienia wszystkie funkcje bibliotek dla FT22x jak i FT20x nazwałem tak samo. Różnią się tylko sufiksem:

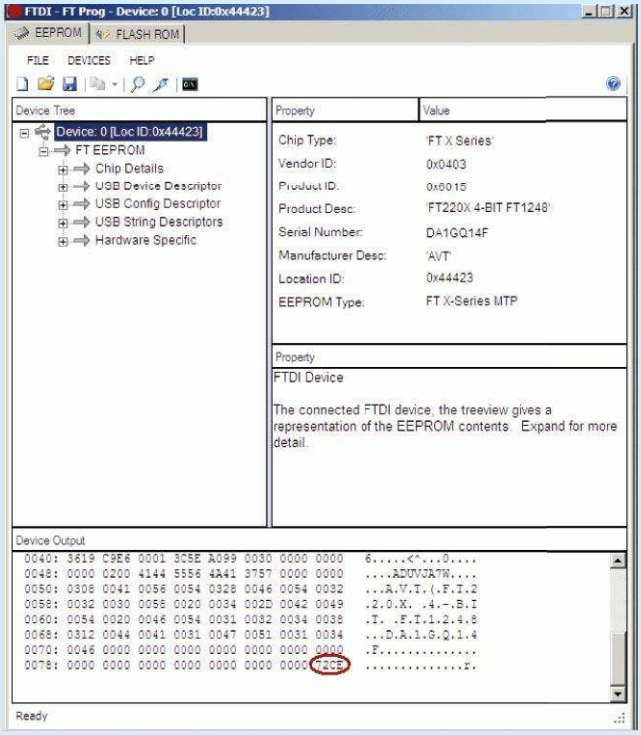
ReadMtpFT201 – ReadMtpFT22x

PutCharFT201 – PutCharFT22x

Parametry funkcji są takie same tyle, że niektóre nie będą istniały. Na przykład, nie ma funkcji AvailableFT22x dla FT22x czy dla FT201: ReadModemFT201, SetModemFT201. Wracając do pamięci MTP. Jej obszar jest podzielony na kilka części – podział pokazano na **rysunku 3**. Obszary oznaczone zielonym kolorem można używać do swoich celów, przy czym „User Area 1" jest widoczny w aplikacji FT_PROG. Obszary „czerwone" są chronione sumą kontrolną. Sumę tę oblicza funkcja „CalculateMtpCrc" w bibliotece „FTDI.c". Sumę kontrolną można odczytać także FT_PROG-iem, jak pokazano na **rysunku 4**.

Listing 7. Sprawdzenie statusu układy FT201

```
byte ReadStFT201()
{
    byte st, repeat=FT201REPEAT;
    StFT201 = -1;
    do
    {
        //----- START -----//
        st = TWI_Start();
        if (st != 8 && st != 0x10) continue;
        // General adres
        st = TWI_Write(0);
        if (st != 0x18)
        {
            TWI_Exit();
            continue;
        }
    //str.22  „AN_255_USB to I2C Example using the FT232H and
FT201X devices"
    //cmd status
    st = TWI_Write(0x16);
    if (st != 0x28)
    {
        TWI_Exit();
        continue;
    }
    //----- START -----//
    st = TWI_Start();
    if (st != 8 && st != 0x10) continue;
    st = TWI_Write(FT201RD);
    if (st != 0x40)
    {
        TWI_Exit();
        continue;
    }
    // Odczytaj ostatni bajt
    StFT201 = TWI_Read( false );
    if (TWI_Status != 0x58 )
    {
        TWI_Exit();
        continue;
    }
//0x00 Suspended
//0x01 Default
//0x02 Addressed
//0x03 Configured
    //----- STOP -----//
    st = TWI_Stop();
    } while( repeat-- );
    return( StFT201 );
}
```

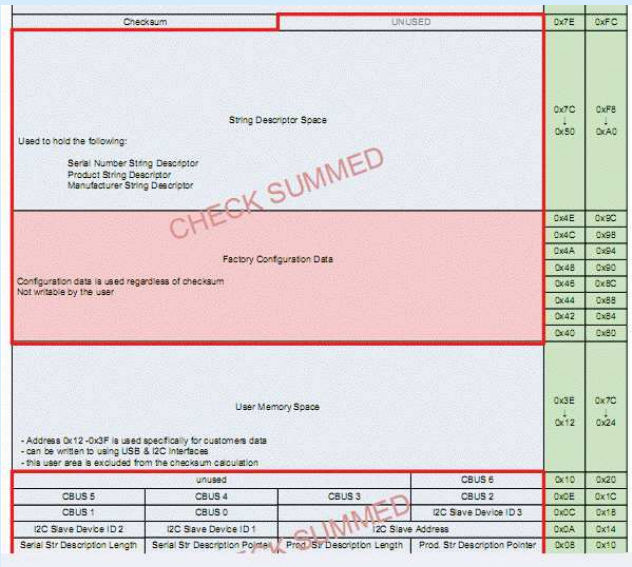


Rysunek 4. Odczyt sumy kontrolnej za pomocą FTProg

Dokładniej omówimy obszar konfiguracji (czerwony). Jego mapę pokazano na **rysunku 5**. Można w nim zapisywać i odczytywać informacje o VID, PID, funkcjach CBUS, poborze prądu, itp. Oczywiście, zmiana adresu I²C nie odniesie skutku dla układów komunikujących się po SPI, podobnie jak konfiguracja nieistniejącego CBUS. Obszar od 0xA0 do 0xF8 (w słowach 0x50..0x7C) przechowuje informacje o nazwie interfejsu, producencie, numerze

Listing 8. Odczytanie identyfikatora układu FT201

```
byte ReadIdFT201()
{
    byte st;
    IdFT201[0] = IdFT201[1] = IdFT201[2] = -1;
    //----- START -----//
    st = TWI_Start();
    if (st != 8 && st != 0x10) return;
    //str.22 „AN_255_USB to I2C Example using the FT232H and
FT201X devices”
    //cmd read ID
    st = TWI_Write( 0xF8/*0x7C*/ );
    if (st != 0x18)
    {
        TWI_Exit();
        return;
    }
    st = TWI_Write(FT201RD);
    if (st != 0x28)
    {
        TWI_Exit();
        return;
    }
    //----- START -----//
    st = TWI_Start();
    if (st != 8 && st != 0x10) return(st);
    //ID command 0x7C
    st = TWI_Write(0xF9/*0x7c*/);
    if (st != 0x40)
    {
        TWI_Exit();
        return;
    }
    IdFT201[0] = TWI_Read(true);
    if (TWI_Status != 0x50 )
    {
        TWI_Exit();
        return;
    }
    IdFT201[1] = TWI_Read(true);
    if (TWI_Status != 0x50)
    {
        TWI_Exit();
        return;
    }
    // Odczytaj ostatni bajt
    IdFT201[2] = TWI_Read(false);
    if (TWI_Status != 0x58)
    {
        TWI_Exit();
        return;
    }
//ID[0] - pewnie vid pid jeśli różny od domyślnego
//ID[1] -
//ID[2] - wartośc ustawiana w FT_PROG
    //----- STOP -----//
    st = TWI_Stop();
}
```

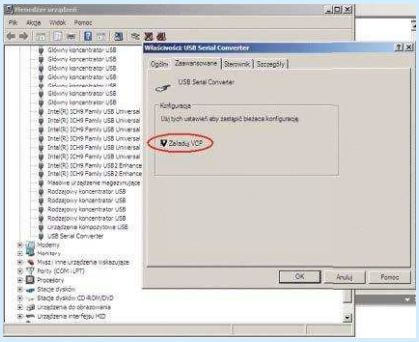


Rysunek 5. Obszar danych konfiguracji w pamięci MTP

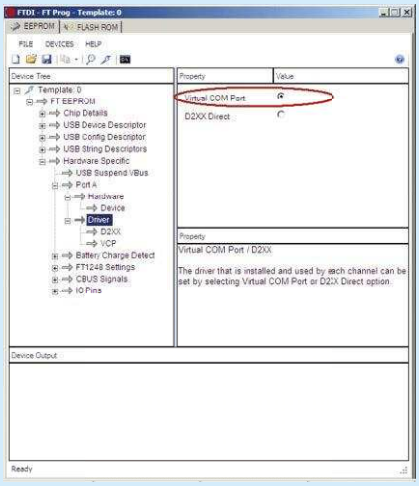
seryjnym. Wskaźniki do tego obszaru zawierają bajty 0x0E..0x13. Jak później pokażę, wskaźników tych w wielu wypadkach nie trzeba obliczać. Dość istotne znaczenie ma pierwszy bajt. W nim jest zawarta informacja między innymi o tym, czy ładować biblioteki VCP, czy nie. W przeciwieństwie do układów z rodziny FT232, FT23x, w mostkach IIC/SPI domyślnym ustawieniem jest nieładowanie VCP

Listing 9. Ustawienie bitu odpowiedzialnego za ładowanie VCP

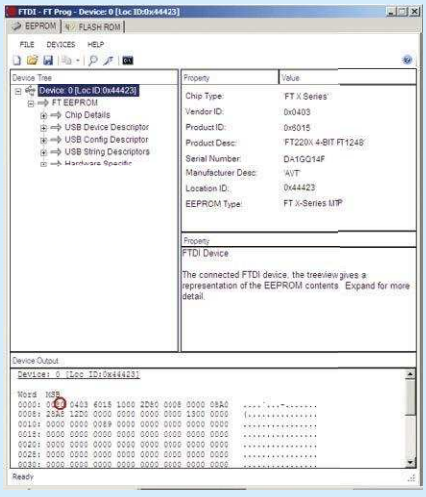
```
ReadMtpFT22x( 0, 256, mtp );
crc = CalculateMtpCrc( (word*)mtp );
crcL=crc;
if ((crcL==mtp[0xFE]) && (crc>>8==mtp[0xFF]))
{
    //jeśli CRC ok to zapis
    WriteMtpFT22x( 0x24, data );
    //
    mtp[0] |= 0x80;  // Ustaw ładowanie VCP
    crc = CalculateMtpCrc( (word*)mtp );
    mtp[0xFE] = crc; mtp[0xFF] = crc >> 8;
    WriteMtpFT22x( 0, mtp[0] );
    WriteMtpFT22x( 0xFE, mtp[0xFE] );
    WriteMtpFT22x( 0xFF, mtp[0xFF] );
}
```



Rysunek 6. Menedżer Urządzeń Windows – włączenie ładowania VCP



Rysunek 7. FTProg – włączenie ładowania VCP



Rysunek 8. Zaznaczenie bitu odpowiedzialnego za ładowanie VCP

(ładowanie D2XX). Jest to o tyle istotne, że taki układ nie będzie widziany jako wirtualny COM i komunikacja z nim będzie możliwa tylko przez biblioteki D2XX. Można to zmienić za pomocą Menedżera Urządzeń zaznaczając opcję „załaduj VCP” jak na **rysunku 6** lub lepiej – z użyciem FTProg, jak na **rysunku 7**. Dlaczego sugeruję robić to z poziomu FT_PROG? Otóż, jeśli zrobimy to w „Menedżerze Urządzeń”, to po zmianie deskryptora i enumeracji konieczne będzie powtórzenie operacji w „Menedżerze Urządzeń”. Jeśli zrobimy to w FT_PROG, to ta „wątpliwa przyjemność” ominie nas. Oczywiście, lepszym rozwiązaniem będzie zmiana bitu odpowiedzialnego za tę funkcje z poziomu mikrokontrolera. Jest siódmy bit (licząc od zera) pierwszego bajtu obszaru MTP (**rysunek 8**). Można to również wykonać za pomocą programu w sposób pokazany na **listingu 9**.

Sławomir Skrzyński, EP