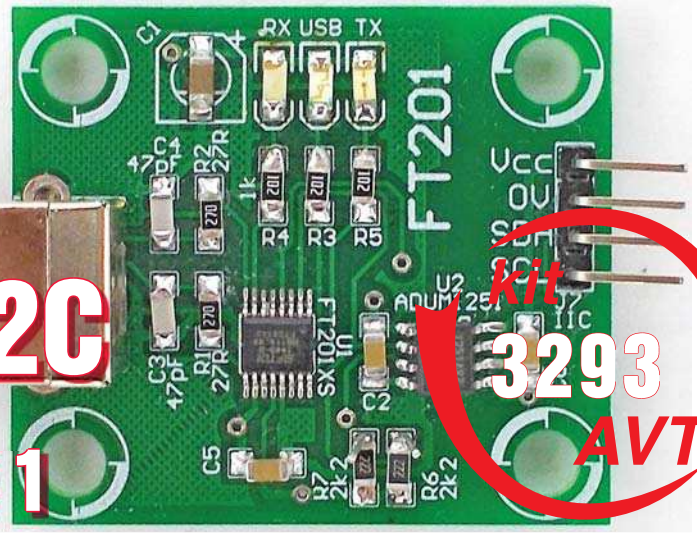
Izolowany galwanicznie mostek USB-I2C

część 1



Prezentowany w artykule konwerter zapewnia izolację galwaniczną między interfejsem USB a mikrokontrolerem. Komunikacja z konwerterem odbywa się nie przez UART, tylko z wykorzystaniem I2C. Interfejs I2C w porównaniu z UART ma wiele zalet.

Opisywane urządzenie służy do galwanicznej izolacji interfejsu USB od mikrokontrolera. Istnieją gotowe izolatory USB, na przykład ADuM1460, ale do tanich nie należą. Zdecydowanie taniej można zbudować izolator z wykorzystaniem układów FT201 i ADuM1250. Układ FT201 jest mostkiem USB-I2C i w stosunku do mostka z interfejsem UART ma szereg zalet, zwłaszcza przy współpracy z AVR, PIC czy zapomnianym już 8051:

– AVR mają mało UART. Jest to szczególnie odczuwalne w przypadku ArduinoUNO. Niektóre AVR mają 2 UART, nieliczne Mega po cztery, ale są one montowane w obudowach 100pin.
– Zegar AVRmega/tiny musi być stabilizowany rezonatorem kwarcowym. Wbudowany RC ma zbyt małą stabilność, aby wykorzystać go z UART.
– Wbudowane w mostki USB-UART FIFO nie spełnia swojej roli. Mostki mają FIFO, ale jeżeli znaki, które przyszły z USB, są wysyłane do UART, czy uC tego chce, czy nie. Ten problem najbardziej odczuwalny jest w Arduino z AVRmega/tiny, w którym podczas komunikacji z WS2812 czy 1-Wire najczęściej zawieszane są przerwania.
– Kontrola przepływu wymaga dodatkowych GPIO uC oraz implementacji jej po stronie HOST-a, co nie zawsze jest możliwe, przykładowo gdy nie mamy kodów źródłowych HOST-a.
– Używając UART, nie można stwierdzić, czy USB HOST jest przyłączony, czy nie. Do tego trzeba zaangażować kolejne linie mikrokontrolera.
– Konfigurację mostka, czyli VID, PID, desktyptor, funkcje GPIO, itd. można przeprowadzić (o ile w ogóle można, bo w np. CP2101 nie) tylko z poziomu komputera odpowiednią aplikacją. Dla układów FTDI jest to FT_PROG. Nie można tego zrobić z poziomu uC.

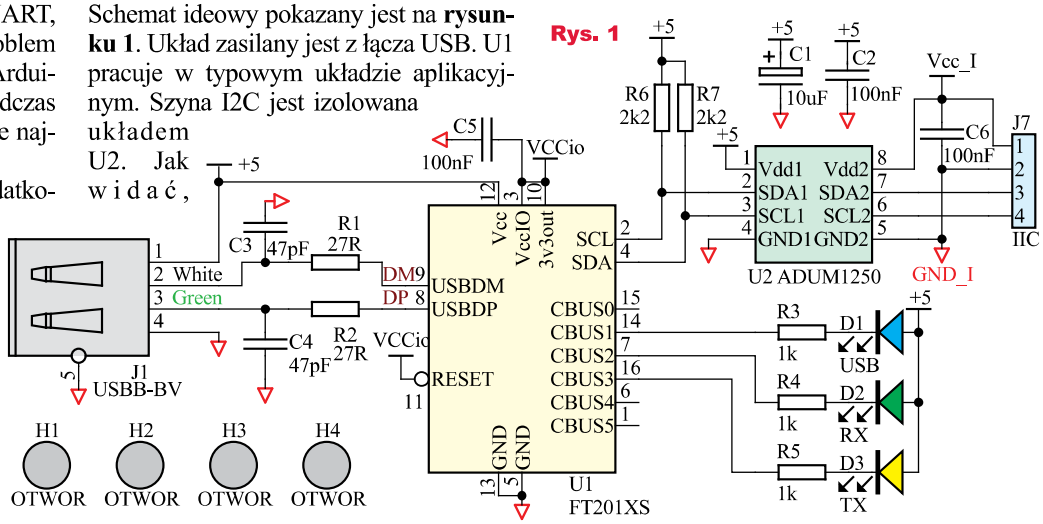
Wszystkich wyżej wymienionych wad pozbawiony jest układ FT201:

– Komunikacja interfejsem I2C do 3,4MHz (w projekcie konwertera prędkość ogranicza ADuM1250 do 1MHz).
– I2C akceptuje logikę 5V.
– Prędkość komunikacji USB 1Mb/s.
– Dwa bufory FIFO 512 bajtów.
– Wszystkie opcje konfiguracji programem FT_PROG dostępne z poziomu interfejsu I2C.
– Ponad 1kB EEPROM do dyspozycji użytkownika.
– 5 konfigurowanych linii GPIO, które między innymi mogą sygnalizować nieodebrane znaki w FIFO, wolne miejsce w FIFO nadawczym, sterowanie LED-ami sygnalizacyjnymi.

Opis układu

Schemat ideowy pokazany jest na **rysunku 1**. Układ zasilany jest z łącza USB. U1 pracuje w typowym układzie aplikacyjnym. Szyna I2C jest izolowana układem U2. Jak widać, budowa sprzętowa jest banalna, cała moc leży w oprogramowaniu. Zanim przystąpię do opisu oprogramowania, wyjaśnię funkcjonowanie FIFO w mostkach USB. W przypadku mostków z UART, dane przychodzące po USB są zapisywane w FIFO, skąd są wysyłane przez UART. Nie znam sposobu, aby zatrzymać wysyłanie danych z FTDI. Zmiana stanu linii CTS czy DTR powoduje tylko zmianę stanu tych wirtualnych linii dostępnych w HOST przez API. HOST może reagować na stan CTS/RTS, ale to, co już jest w buforze układu FTDI, musi zostać wysłane. Z tego powodu reakcja na zmianę CTS/RTS nie jest natychmiastowa i w przypadku układów FTDI w najgorszym przypadku mikrokontroler może jeszcze otrzymać 512 znaków od czasu zmiany stanu linii CTS/RTS. Ponadto, nie zawsze program będzie reagował na stan owych linii. Jeśli nie mamy kodów źródłowych, nic z tym się nie da zrobić. Arduino nader często blokuje przerwania. W przypadku transmisji do LED WS2812 taka blokada może trwać kilkadziesiąt milisekund, a przy prędkości 115200, blo-

Rys. 1



kada na dłużej niż ok. 173µs spowoduje gubienie znaków. W przypadku 921600 wystarczy ok. 22µs.

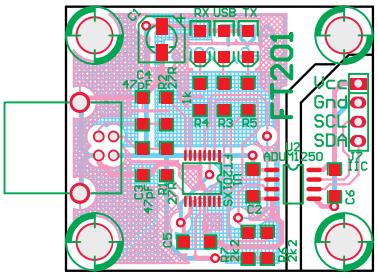
Z FT201 jest inaczej. Dane przychodzące po USB są tak jak i w przypadku mostków z UART zapisywane w FIFO, ale FIFO będzie odczytane dopiero wtedy, gdy zrobi to mikrokontroler, bo FT201 jest układem slave i sam z siebie nie wyśle danych na I2C. Pozwala to na zdecydowanie dłuższy czas blokować przerwania mikrokontrolera, a nawet obsłużyć komunikację I2C bez użycia przerwań.

Montaż i uruchomienie

Układ można zmontować na płytce drukowanej, której projekt pokazany jest na **rysunku 2**. Układ montujemy standardowo, zaczynając od elementów najmniejszych, a kończąc na największych.

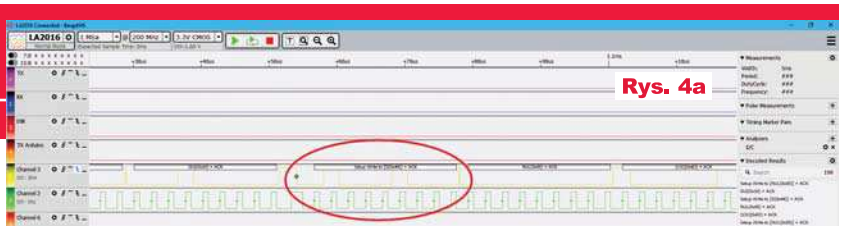
Fotografia wstępna oraz **fotografia 1** pokazują model. Układ zmontowany prawidłowo ze sprawnych elementów powinien od razu pracować.

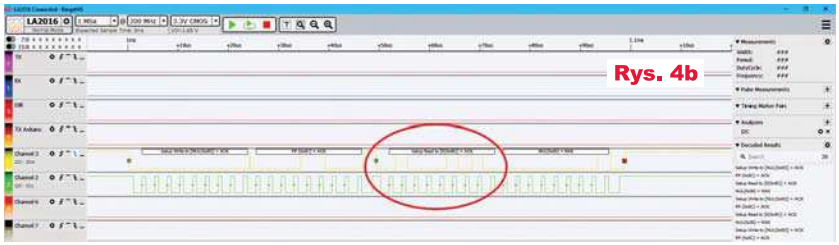
Obsługa programowa. Podstawowa funkcjonalność związana z transmisją danych jest osiągalna w zadziwiająco prosty sposób. Aby wysłać dane po USB, wystarczy zaadresować układ slave o adresie 0x22 (0x44) do zapisu. Adres 0x22 jest domyślnym adresem układu FT201, można go zmienić programem FT_PROG (**rysunek 3**) lub modyfikując obszar pamięci MTP przez mikrokontroler, o czym później. ***Uwaga! Wszystkie rysunki – zrzuty z artykułu o dużej rozdzielczości są dostępne w Elportalu wśród materiałów dodatkowych do tego numeru EdW.*** Adres w programie FT_PROG wpisujemy w postaci liczby szesnastkowej 7-bit. Trzeba o tym pamiętać, bo łatwo o pomyłkę, ponieważ adres 0x22 w analizatorze czy na oscyloskopie jest reprezentowany w postaci liczby 0x44 przy zapisie (**rysunek 4a**) i 0x45 przy odczycie (**rysunek 4b**). Programy operują na różnych adresach, przykładowo Arduino używa adresowania 7-bitowego, co oznacza, że domyślnym adresem FT201 jest 0x22, natomiast HAL STM32 posługuje się adresem 8-bitowym, więc



Rys. 2

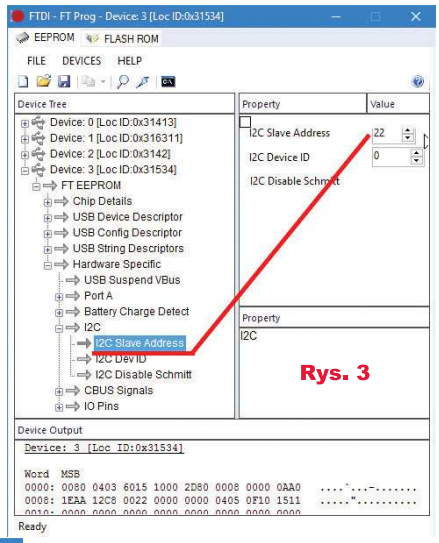
co widać w kodach źródłowych, należy posługiwać się adresem 0x44. Po zaadresowaniu slave brak potwierdzenia ACK świadczy o zapełnionym buforze FIFO.


Rys. 4a

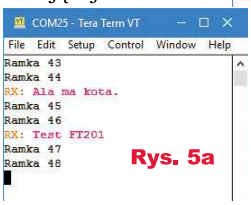

Rys. 4b

Odbiór jest równie prosty. Po zaadresowaniu slave do odczytu należy sprawdzić sygnał ACK. Jeśli go nie ma, to w FIFO nie ma znaku do odczytu, jeśli jest, w FIFO znajduje się co najmniej jeden znak. Na **listingu 1a** znajduje się fragment programu wysyłania i odbioru danych do/z FT201 napisany dla STM32L412, natomiast **listing 1b** przedstawia program dla Arduino. ***Uwaga! Wszystkie listingi do tego artykułu są dostępne w Elportalu wśród materiałów dodatkowych do tego numeru EdW.***

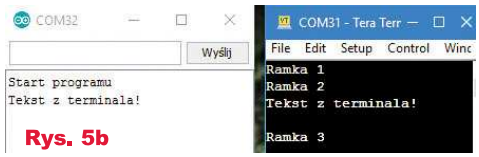
Program wysyła co dwie sekundy napis „Ramka" oraz jej numer i odsyła przychodzące dane, poprzedzając je tekstem „RX:" – **rysunek 5a**. Program dla Arduino nie odsyła tekstów na terminal, tylko wyświetla w monitorze portu szeregowego – **rysunek 5b**. Programy w pełnej wersji dostępne są też w materiałach dodatkowych na Elportalu.



Rys. 3



Rys. 5a

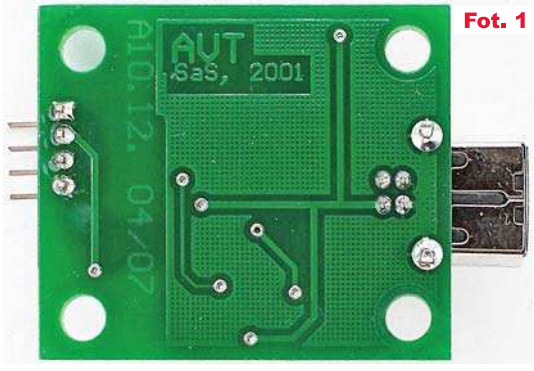


Rys. 5b

Czas podnieść poprzeczkę i wykorzystać zaawansowane możliwości układu. Jak odczytać status układu czy liczbę bajtów zgromadzonych w FIFO? Adresowanie i odczyt układu nie mają sensu, bo zwraca on zawartość FIFO. Aby nie rezerwować kolejnego adresu, konstruktorzy zdecydowali się na wykorzystanie adresu broadcastowego. Ze względu na to, że na adres ten reagują wszystkie układy na magistrali, adres broadcastowy nie służy do odczytu/zapisu rejestrów układu FT201, tylko do wysyłania komend. Aby odczytać, ile bajtów zgromadzono w FIFO, należy:

– Wygenerować START.
– Zaadresować slave 0 (broadcast) do zapisu.
– Wysłać komendę, w przypadku odczytu liczby bajtów w FIFO 0x0C.
– Wygenerować ponowny START. **Nie może to być STOP-START, musi być ponowny start!**
– Zaadresować FT201 do odczytu.
– Odczytać bajt bez ACK. W bajcie zawarta będzie informacja o liczbie bajtów w FIFO.
– Wygenerować STOP.

Rodzi się pytanie, jak jednym bajtem odczytać liczbę 9-bit? FIFO ma 512 bajtów, aby więc poinformować o liczbie bajtów, potrzebne są dwa bajty. Przyznam, że nie wiem, w nocie katalogowej też nic na ten temat nie znalazłem. Pisząc artykuł, opierałem się na przykładzie z noty („AN_255_USB to I2C Example using the FT232H and FT201X devices", strona 25), a wynikiem jest **listing 2**.



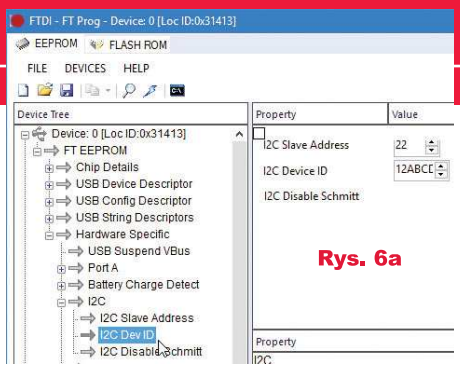
Fot. 1

Wszystko na to wskazuje, że gdy danych będzie więcej niż 255, FT201 zwróci wartość 255. Po odczytaniu danych z FIFO, kolejny odczyt liczby bajtów w FIFO zwróci resztę, która została. Na **listingu 3a** pokazana jest funkcja odczytująca liczbę bajtów w FIFO dla STM32F411 i zmodyfikowana funkcja odsyłająca dane.

Zdziwić może, dlaczego nie użyłem HAL-a? Czyżby nie istniała funkcja, która wysyła ponowny START? Istnieje (`HAL_I2C_Mem_Read`), ale niestety, twórcy HAL nie przewidzieli, że adres do zapisu może być inny niż do odczytu. Ze względu na to, że zrozumienie HAL STM32 jest zadaniem trudnym, prościej było napisać ten fragment, posługując się rejestrami. Ta sama funkcja dla Arduino jest na **listingu 3b**.

Tu też skorzystałem z rejestrów, bo biblioteki Arduino są niedopracowane. Standardowe wcale nie pozwalają na wygenerowanie ponownego startu (HAL STM32 tylko wtedy, gdy adresy slave się różnią). To bardzo dziwne postępowanie, zwłaszcza że wiele układów korzysta z tej funkcjonalności. Ponadto, bufor nadawczy jest ograniczony do 32 bajtów. Bez modyfikacji bibliotek zwiększenie bufora nadawczego zwiększy także bufor odbiorczy. W konsekwencji, gdyby chcieć wykorzystać maksymalny dostępny rozmiar dla FT201 o wielkości 512 bajtów, zwiększenie bufora dla I2C spowodowałoby zużycie połowy RAM dostępnej w ArduinoUNO!

Trochę odejdę od tematu głównego i opiszę problem, z którym już kiedyś się spotkałem i dał mi się we znaki podczas pisania oprogramowania dla FT201. W magistrali I2C, w specyficznych warunkach może dojść do sytuacji, że slave zablokuje magistralę. Może tak się stać wtedy, gdy mikrokontroler zostanie zresetowany w momencie, gdy slave wystawia zero na magistrali danych. W takiej sytuacji, po resecie, nie można poprawnie przeprowadzić komunikacji po i2C, ponieważ master stwierdza, iż magistrala jest zajęta przez inny master. Szanse na zaistnienie takiego zdarzenia są tym większe, im częściej odczytuje się dane ze slave. W przypadku FT201, gdy nie używamy przerwań od FIFO, odpytywanie jest bardzo częste (dobry powód, aby jednak korzystać z przerwań). Jak wyjść z takiej patowej sytuacji, gdy slave blokuje magistralę? Najprościej zresetować slave, ale nie każdy układ ma taką możliwość, na przykład FCP8574. Co



Rys. 6a

wtedy? Wystarczy wygenerować dziewięć impulsów na linii SCK, po czym warunek stopu. Niestety nie da się zrobić tego z układu I2C master wbudowanego w mikrokontroler. Należy go wyłączyć i prostym programem wygenerować impulsy oraz STOP. Funkcję taką można także wywołać, gdy nagle tracimy komunikacje z układami slave. Czasem pomaga. Kod takiej funkcji dla STM32 i miejsce jej umieszczenia pokazuje **listing 4a**. To samo dla AVR pokazuje **listing 4b**.

Gdy już mamy pewną obsługę I2C odczytamy status FT201. Operacja przebiega tak samo jak w przypadku odczytu liczby bajtów w FIFO, aby więc nie tworzyć kolejnych funkcji, stworzymy uniwersalną, której argumentem będzie komenda dla układu FT201 jak na **listingu 5a**.

Status jest liczbą z zakresu 0...3 a oznacza:

0x00 Suspended
0x01 Default
0x02 Addressed
0x03 Configured

Jeśli układ został skonfigurowany przez HOST, status zwraca 3. Odczyt statusu dla Arduino na **listingu 5b**.

Kolejną funkcją jest odczyt ID układu. ID można ustawić programem FT_PROG – **rysunek 6a**. ID zawiera trzy bajty, w FT_PROG wprowadza się je w formie liczby szesnastkowej. Aby odczytać ID przez I2C, nie jest używany jak poprzednio adres broadcastowy, tylko używa się dodatkowego adresu układu FT201: 0xF8 (0x7C). Niestety, operacja nie jest standardowa, tak jak i poprzednia z adresem broadcastowym. Aby odczytać ID, należy:

– Wygenerować START.
– Zaadresować slave 0xF8 (0x7C) do zapisu.
– Wysłać adres układu FT201 do odczytu.
– Wygenerować ponowny START (nie może być stop-start musi być ponowny start).
– Zaadresować slave 0xF9 (0x7C) do odczytu.
– Odczytać bajt bez ACK. W bajcie zawarta będzie informacja o liczbie bajtów w FIFO.
- Wygenerować STOP.

W tym celu napisałem funkcję pokazaną na listingu 6a. Odczyt ID na Arduino na **listingu 6b** (tylko w Elportalu).

Te karkołomne sztuczki z adresem broadcastowym i F8 pozwalają na używanie wielu układów na jednej magistrali, a jednocześnie nie trzeba rezerwować osobnych adresów dla danych i komend lub wydłużać transmisji o dodatkowy bajt informujący, czy chcemy operować na danych USB, czy na rejestrach układu FT201.

Na koniec pozostawiłem najciekawszą możliwość FT201: konfigurowanie układu z poziomu mikrokontrolera. **Wszystko, co można zrobić programem FT_PROG, można zrobić także z poziomu mikrokontrolera!** Daje to duże możliwości.

– Pierwsza to fakt, że nie trzeba wgrywać programu do urządzenia dwa razy, raz programu dla mikrokontrolera, za drugim razem konfiguracji dla FTDI.
– Kolejna zaleta to ewentualny upgrade programu. Przykładowo mikrokontroler może pobrać najnowszą wersję programu z Internetu i zaprogramować siebie, ale co z mostkiem USB? Gdy jest to mostek UART, a istnieje konieczność zmiany konfiguracji układu, to mamy sytuację patową. W przypadku FT201 problemu nie ma.
– Kolejny przykład to wymiana uszkodzonego mostka USB. W przypadku standardowych układów trzeba jeszcze wgrać konfigurację, a w przypadku FT201 konfigurację może ustawić mikrokontroler.
– Mikrokontroler może sprawdzić konfigurację, pozwalając zabezpieczyć się przed „grzebaniem" w niej przez osoby postronne.
– MTP i powiązana z nią pamięć EEPROM pozwala na umieszczanie w niej danych, które nie ulegną zniszczeniu po wymianie mikrokontrolera. Pozwala to tworzyć licznik czasu pracy, zabezpieczenia czy konfigurację w postaci dodatkowej kopii przydatnej, gdy zawartość pamięci EEPROM w mikrokontrolerze ulegnie uszkodzeniu.
– Aplikacja może odczytać informacje o urządzeniu, korzystając z EEPROM, przy czym nie ma tu ograniczenia liczby danych do 32, jak w przypadku deskryptora USB.

Ciąg dalszy w następnym numerze.

SaS
sas@elportal.pl