*Dokumentacja projektu kursu  
Układy cyfrowe i systemy wbudowane 2*

**Obsługa transmisji szeregowej w standardzie USB**

**Prowadzący:**Dr inż. Jarosław Sugier

**Autorzy:**  
Sebastian Kuczyński, 209764  
Mikołaj Styś, 209773

**Wrocław, 17.05.2016**

**Spis treści**

[Spis treści 2](#_Toc451195273)

[1. Wstęp 3](#_Toc451195274)

[1.1 Opis teoretyczny 3](#_Toc451195275)

[2. Opis projektu 8](#_Toc451195276)

[2.1. Opisy modułów 10](#_Toc451195277)

[2.1.1 usb\_rx 10](#_Toc451195278)

[2.1.2 sof\_filter 14](#_Toc451195279)

[2.1.3 newline\_fsm 15](#_Toc451195280)

[3. Implementacja programu w układzie FPGA 16](#_Toc451195281)

[4. Instrukcja użytkownika 17](#_Toc451195282)

[5. Podsumowanie 18](#_Toc451195283)

[5.1. Ocena projektu 18](#_Toc451195284)

[5.2. Dalsza praca 19](#_Toc451195285)

[6. Bibliografia 19](#_Toc451195286)

# 1. Wstęp

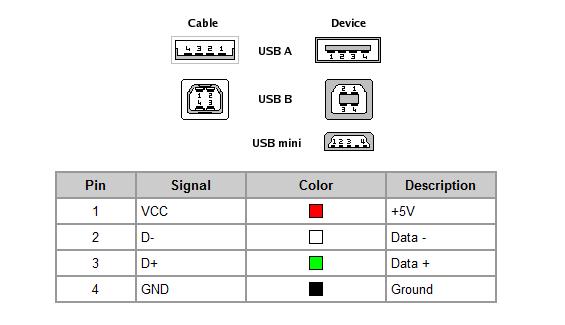
Celem projektu było zaprogramowanie w języku VHDL sprzętowej implementacji sterownika USB standardzie 1.1 Full Speed, za pomocą którego można przesyłać dane. W tym celu otrzymaliśmy możliwość korzystania z komputera z zainstalowanym oprogramowaniem ISE Webpack do tworzenia programu oraz układ FPGA Spartan 3E wraz z podłączonym gniazdem USB typu B do układu.

Układ posiadał dwa gniazda USB. Pierwsze, wbudowane zostało zaprojektowane jedynie   
do przesyłania programu z komputera [4], przez co do wykonania projektu potrzebne   
było wspomniane wcześniej dodatkowe gniazdo USB.

Z uwagi na małą ilość godzin przeznaczoną na dany projekt skoncentrował się on na odbieraniu danych z interfejsu. W tym celu należało zapoznać się ze sposobem elektrycznej reprezentacji danych w łączu oraz sposobem dekodowania i interpretowania otrzymanych danych. Większość danych pochodzi ze specyfikacji USB 1.1 [1].

* 1. **Opis teoretyczny**

Tabele i rysunki w tym podrozdziale pochodzą z kopii dokumentu standardu USB 1.1 [2].

****

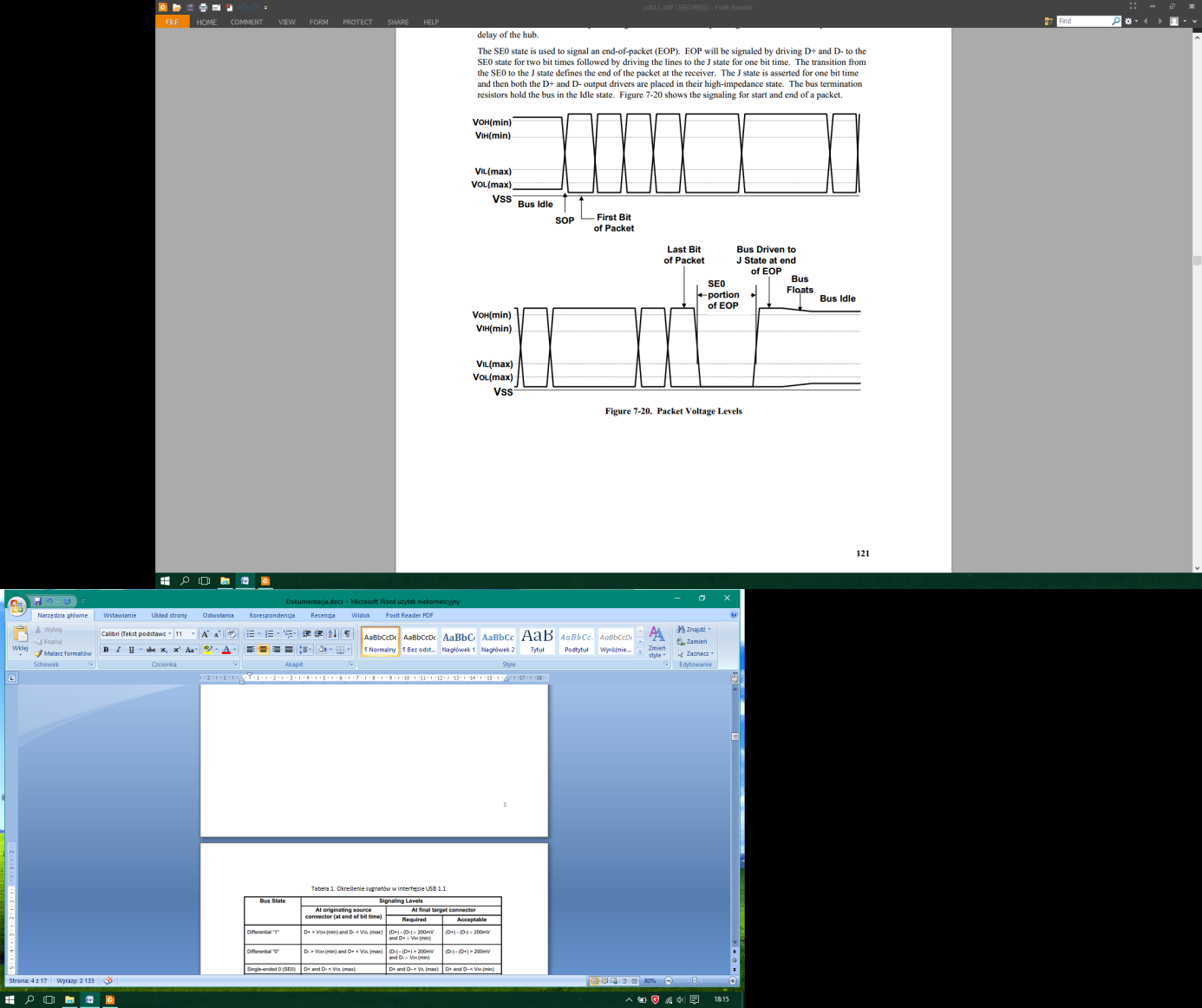
Rysunek 1. Budowa złącz interfejsu USB wraz z opisem pinów [6].

Interfejs USB posiada wiele typów złącz. Użyty przez nas typ to B (po stronie Układu FPGA) i typ A (po stronie komputera). Interfejs posiada 4 przewody – 2 przeznaczone dla zasilania   
oraz 2 do przesyłu danych. Przewody D+ i D- zawierają przeciwny stan sygnału, tzn. gdy D+ jest w stanie wysokim, D- jest w stanie niskim.

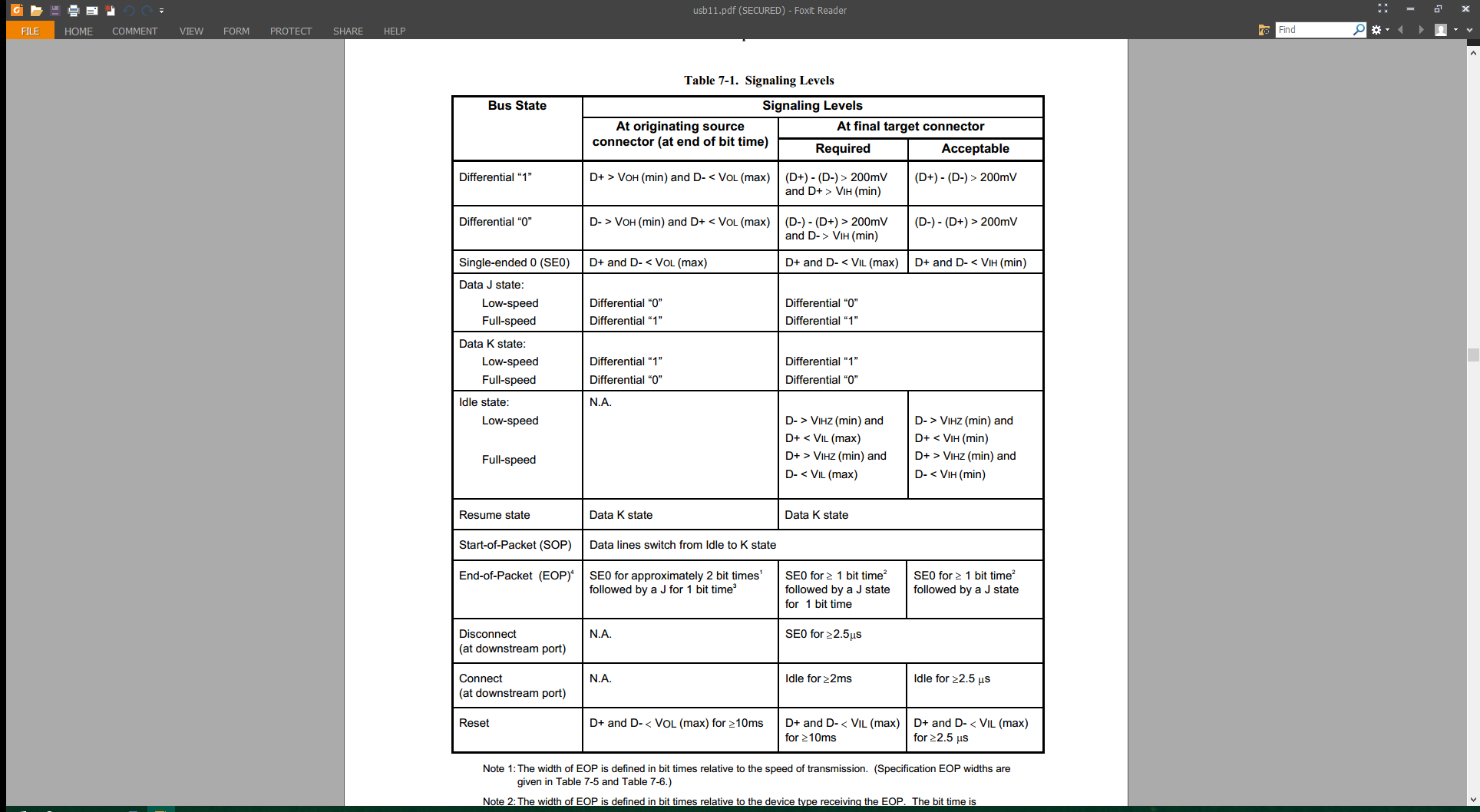
Tabela 1 przedstawia poziomy sygnałów oraz ich znaczenie. W implementowanym standardzie Full-Speed poziom wysoki (tj. stan J) odpowiada wysokiemu poziomowi napięcia – w standardzie Low-Speed poziom wysoki odpowiada niskiemu napięciu. Z tabeli najważniejszymi informacjami   
dla naszego odbiornika są:

* Start-of-Packet – pakiet zaczyna się, gdy po długim czasie stanu wysokiego (sygnał Idle) pojawi się stan niski.
* End-of-Packet – pakiet kończy się, gdy obie linie danych mają niski poziom napięcia przez więcej niż 1 takt. Sygnał ten nazywa się SE0 (Single Edged 0).

Schemat ilustrujący początek i koniec pakietu przedstawiony jest na rysunku 2.



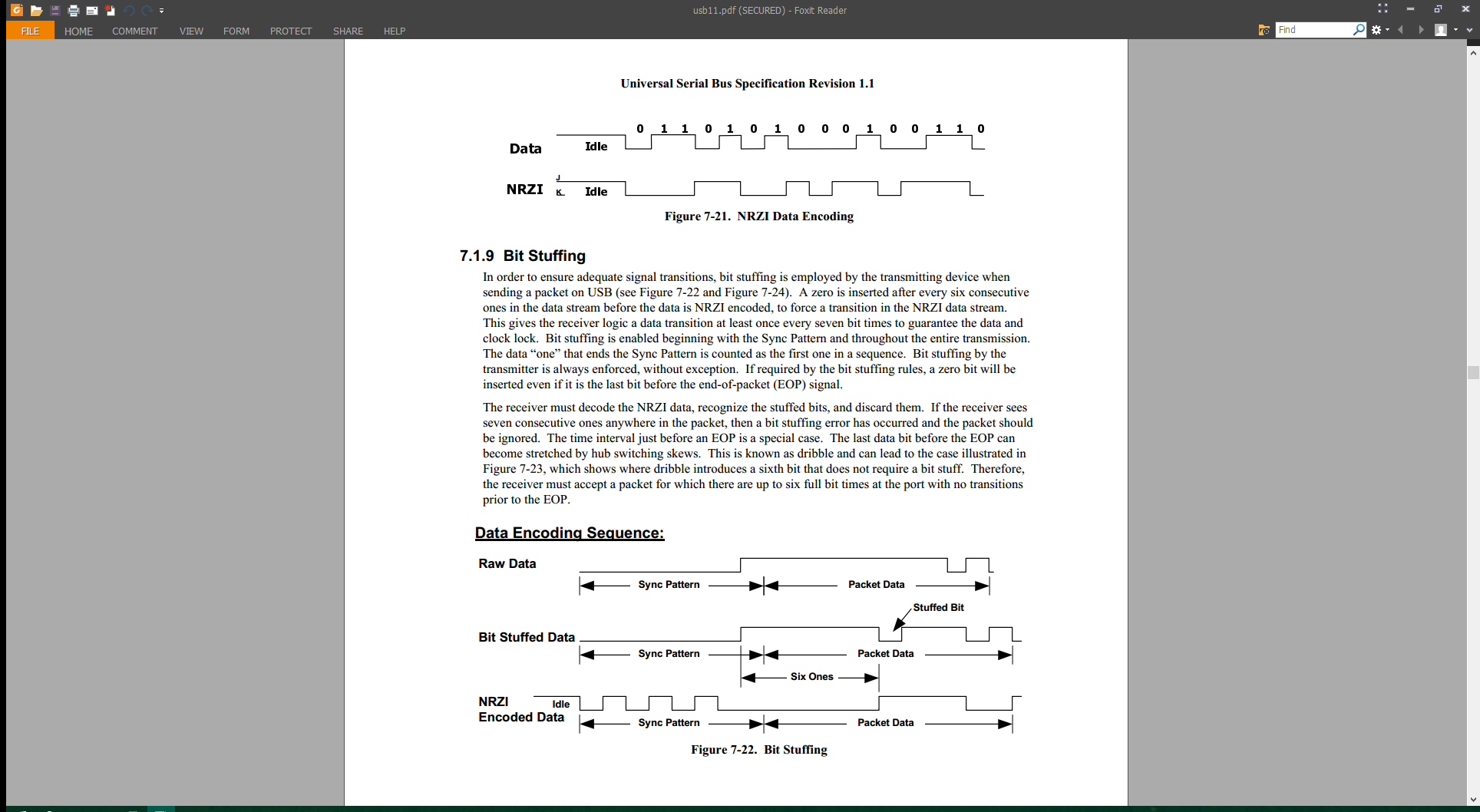
Rysunek 2. Początek i koniec pakietu.

Tabela 1. Określenie sygnałów w interfejsie USB 1.1

Dane przesyłane w interfejsie USB są kodowane za pomocą kodowania NRZI (Non Return to Zero Inverted), gdzie każdy stan logiczny 0 powoduje zmianę stanu sygnału wyjściowego. Przykład kodowania ilustruje rysunek 2. Przy kodowaniu sygnału należy zwrócić na tzw. bit stuffing,   
który ma za zadanie zwiększyć częstość zmiany sygnału co zapobiega rozsynchronizowaniu urządzeń (wychwytywany jest takt interfejsu). Rysunek 3 przedstawia bit stuffing wraz z Sync Pattern,   
czyli nagłówkiem pakietu, który informuje o rozpoczęciu przesyłania pakietu. Sync pattern   
jest zawsze identyczny i w kodowaniu NRZI wynosi 0b01010100 (po odkodowaniu 0x80). Dane przesyłane są w kolejności od najmłodszego bitu do najstarszego.

# 

Rysunek 3. Przykład kodowania NRZI.

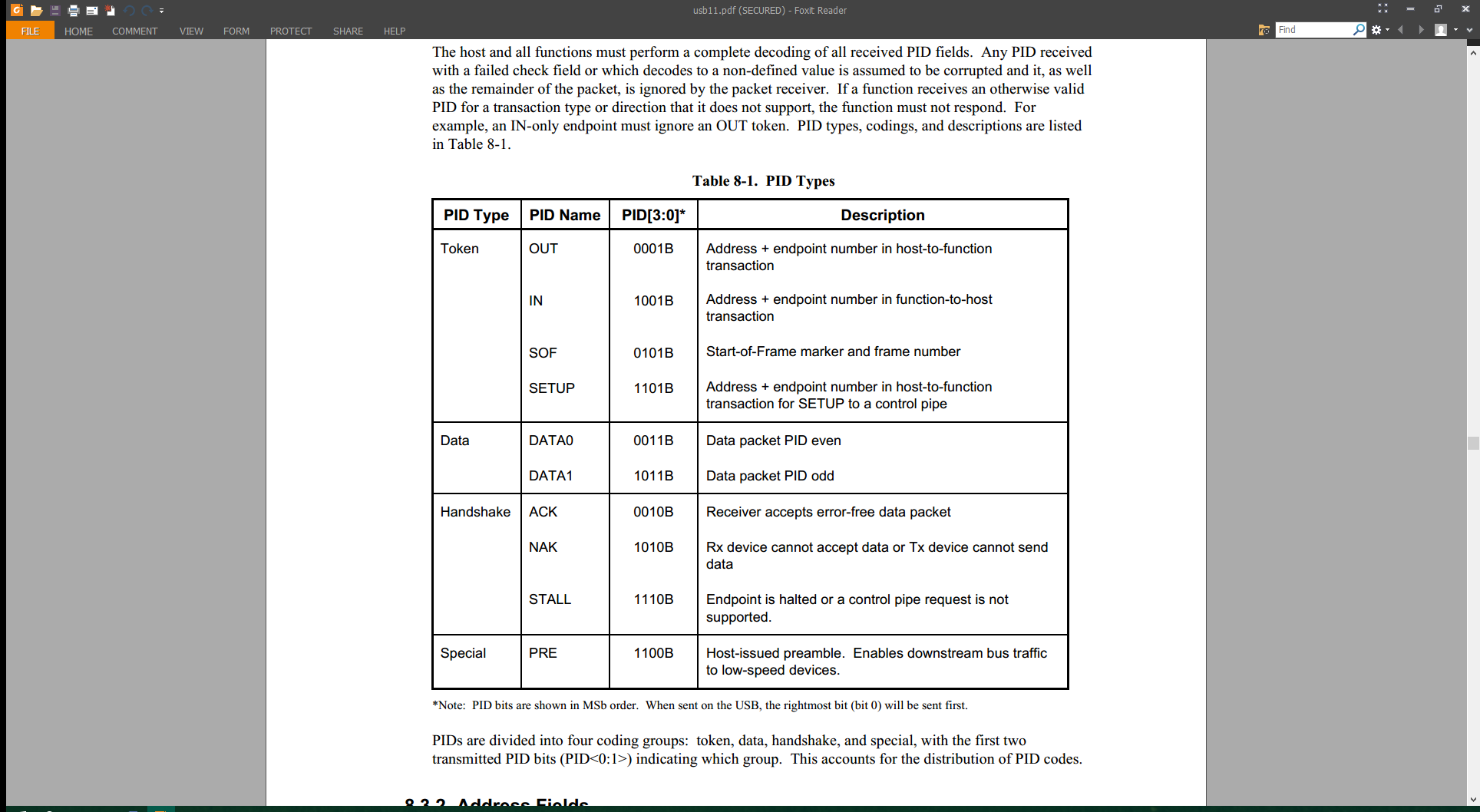


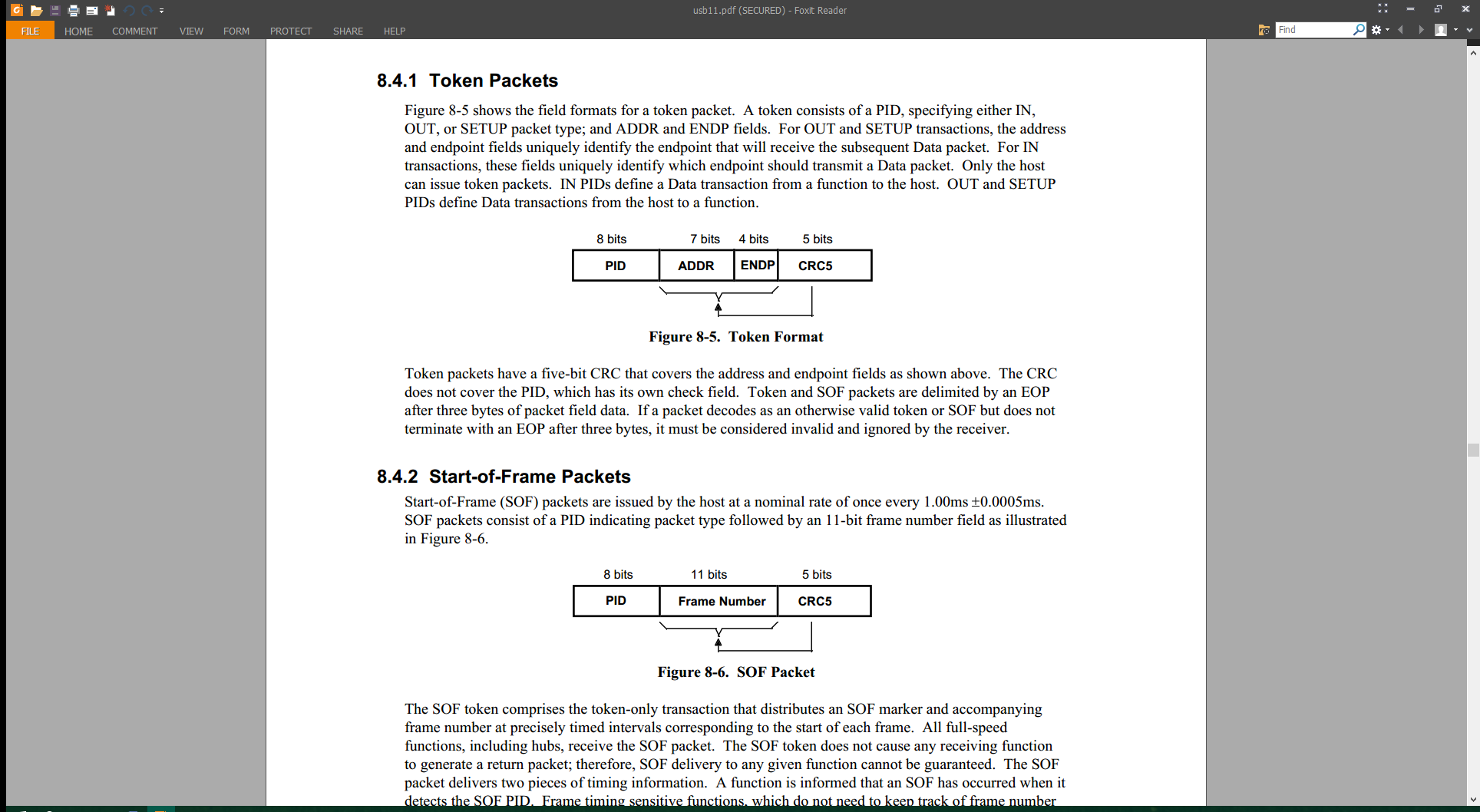
Rysunek 4. Przykład bit stuffing’u oraz sync pattern’u.

Odkodowane dane pakietu przyjmują różną postać w zależności od jego PID. Lista dostępnych PID znajduje się w Tabeli 2. Typy pakietów możliwymi do odczytania za pomocą zbudowanego odbiornika to:

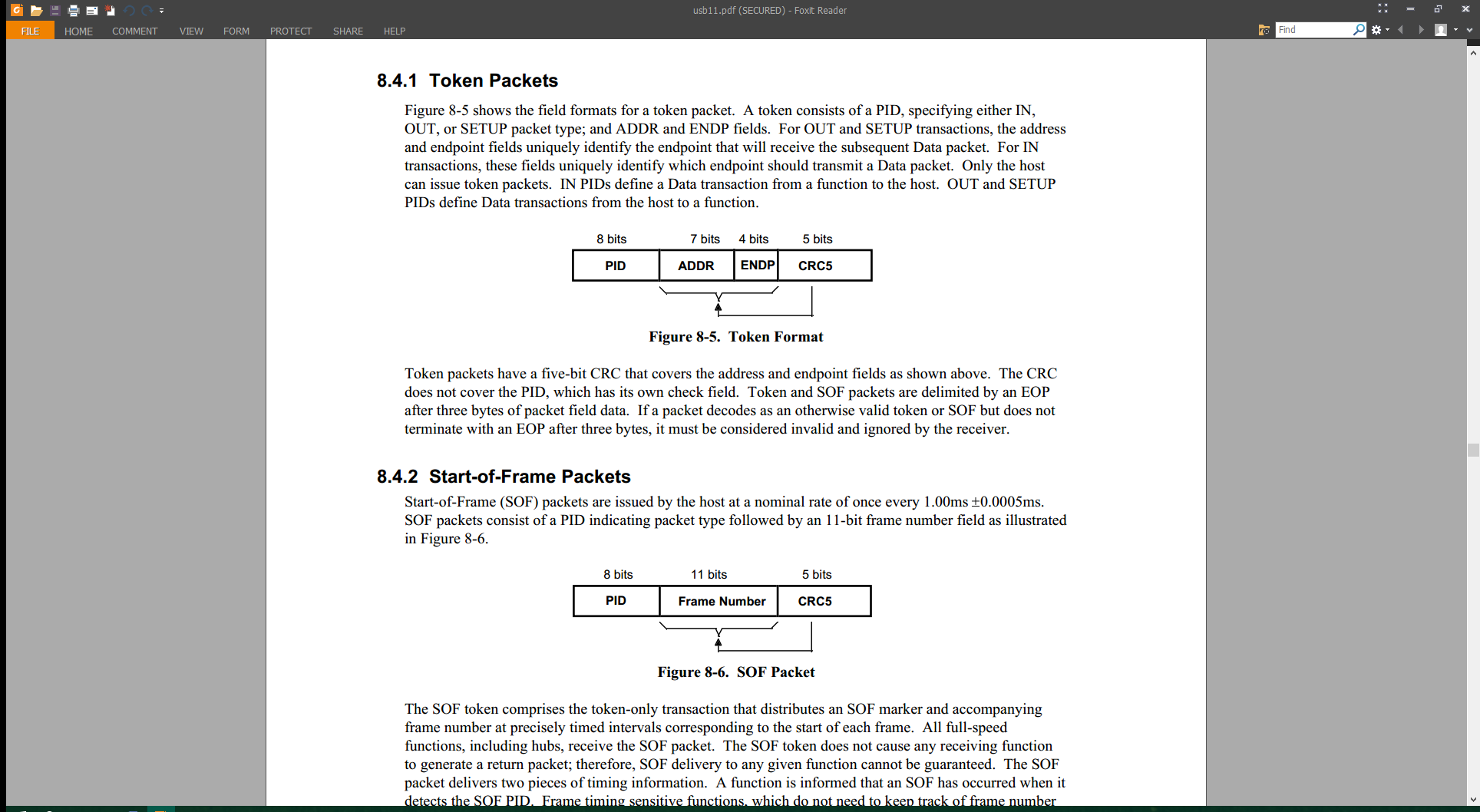
* SETUP – pakiet zawierający informacje do konfiguracji urządzenia (Rysunek 5)
* SOF – Start Of Frame, pakiet wysyłany co 1 ms przez kontroler (Rysunek 6)
* DATA – pakiet z danymi (Rysunek 7)

Tabela 2. Typy PID.

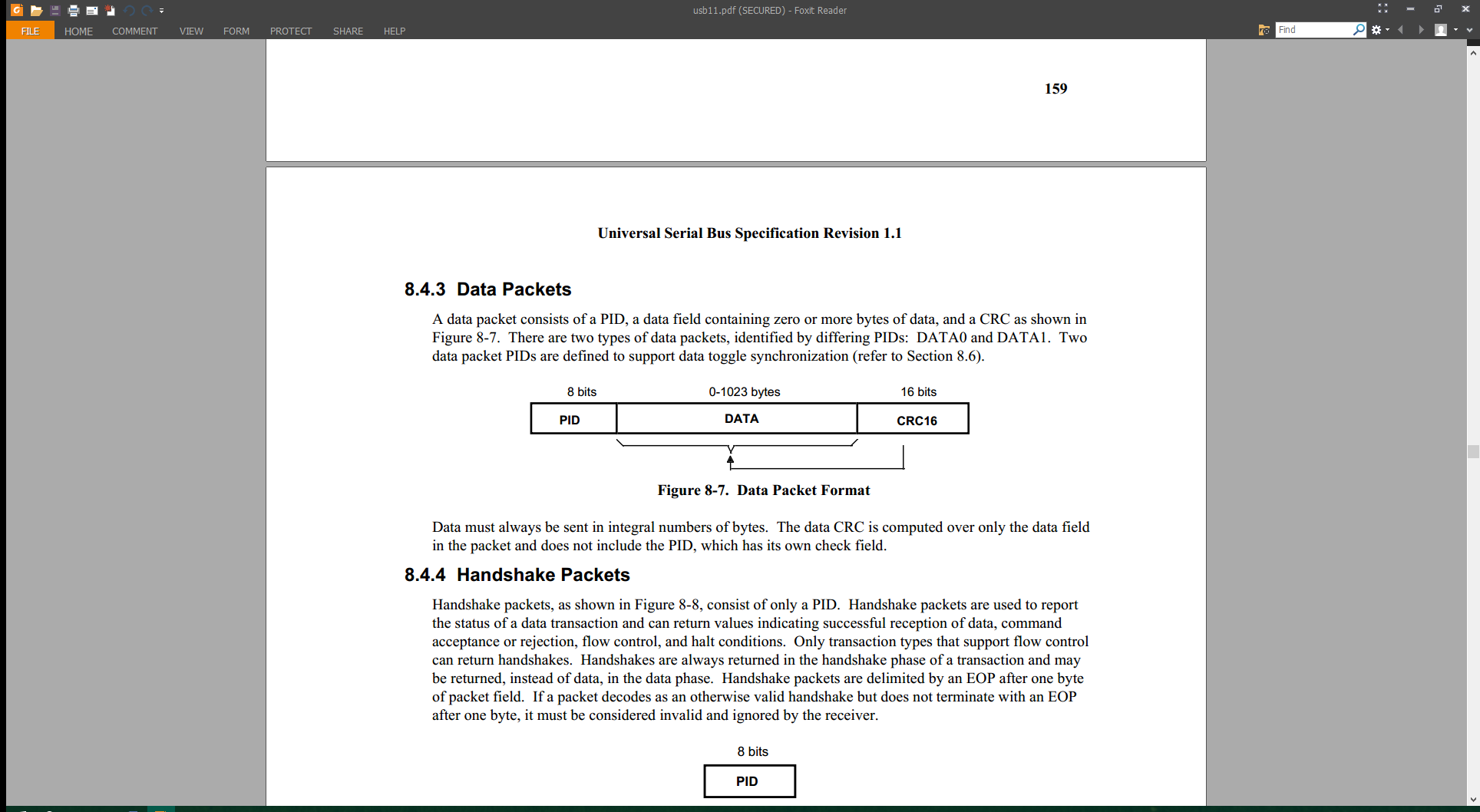




Rysunek 5. Pakiet typu Token (np. SETUP)

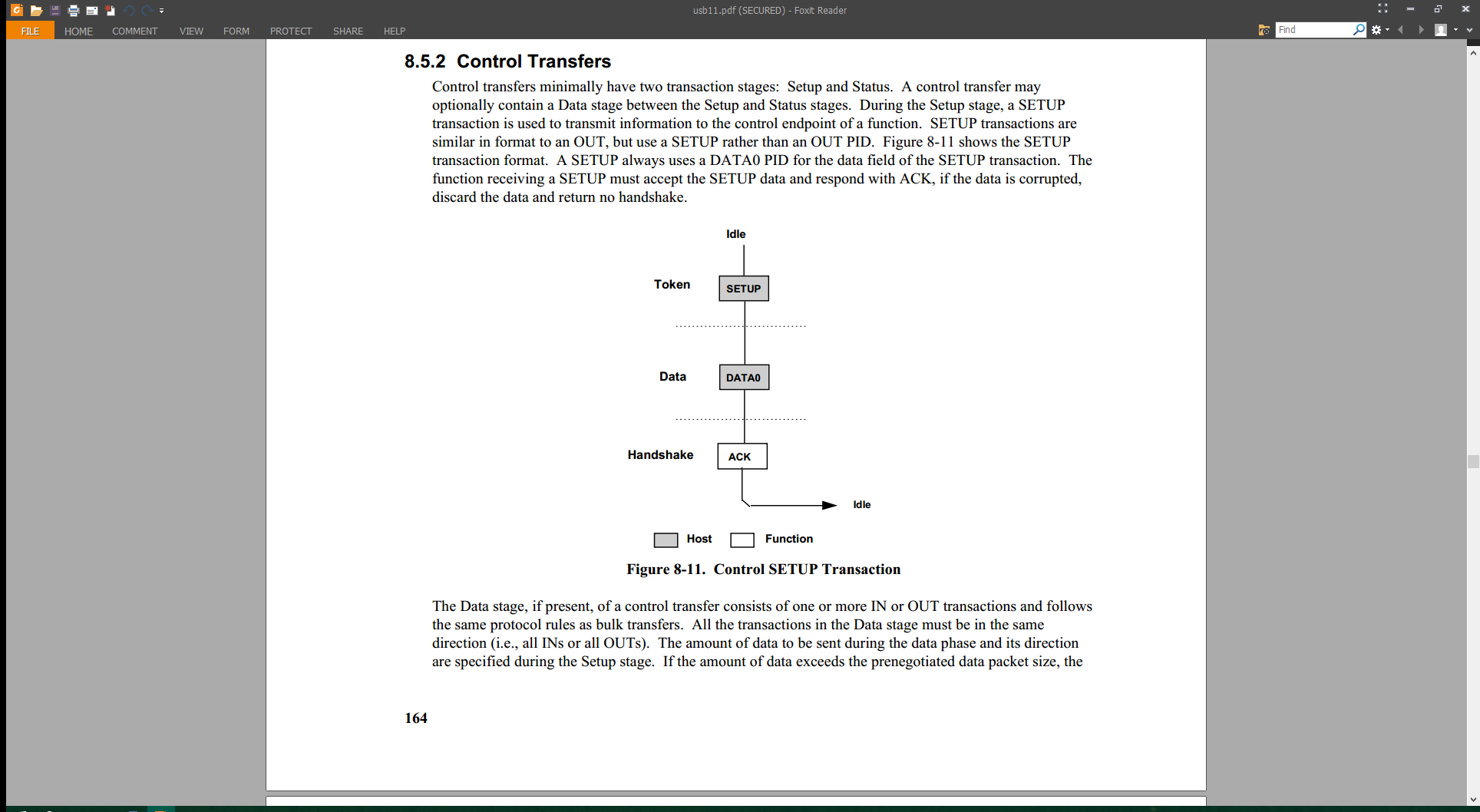


Rysunek 6. Pakiet Start Of Frame



Rysunek 7. Pakiet typu Data.

Aby urządzenie zostało podłączone do kontrolera musi zostać wykonana sekwencja z danymi konfiguracyjnymi wysyłanymi przez kontroler, które powinno zostać zaakceptowane przez urządzenie pakietem ACK. Proces przedstawia rysunek 8.



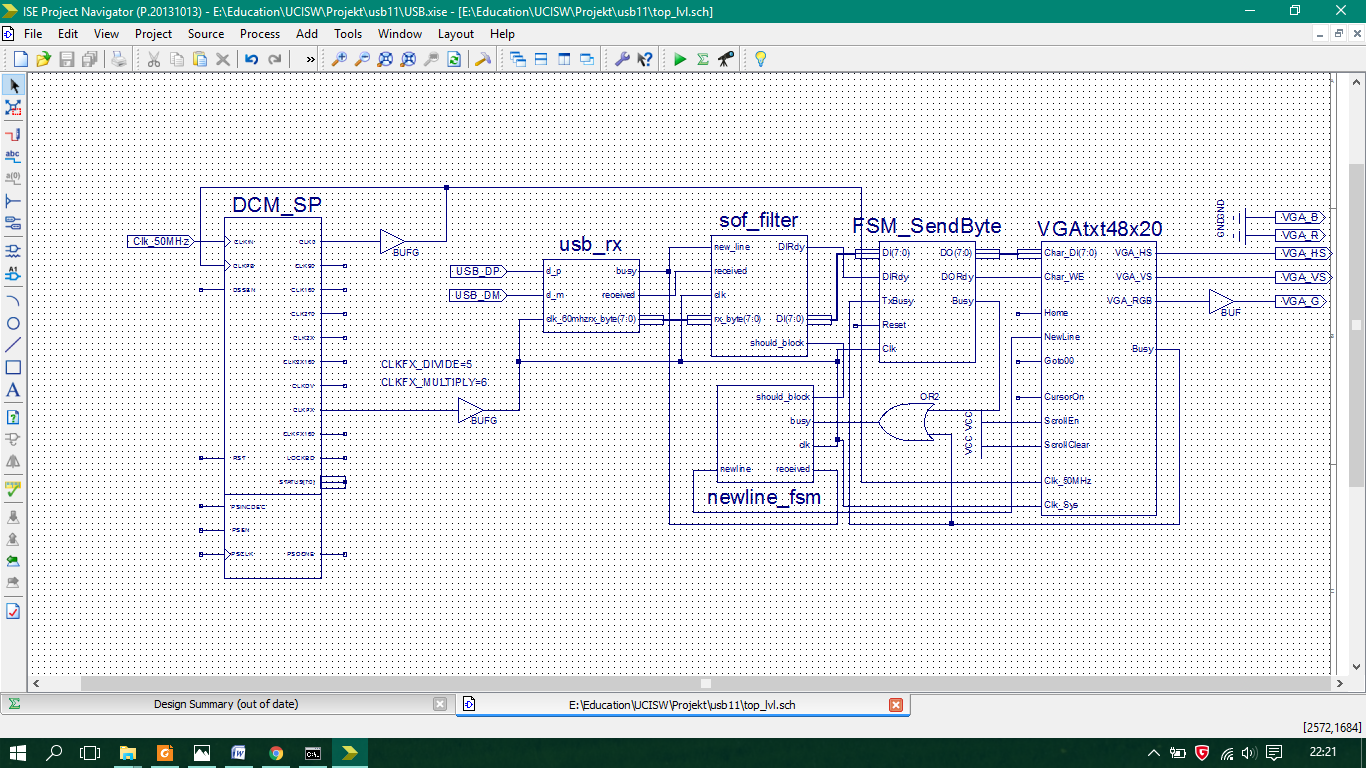
Rysunek 8. Schemat komunikacji podczas podłączania urządzenia.

# 2. Opis projektu

Projekt składał się z głównego schematu **top\_lvl** zawierającego moduły napisane w języku VHDL (Rysunek 9). Moduły znajdujące się na Rysunku 9 to:

* **DCM\_SP** – Digital Clock Manager, odpowiadający za zmianę częstotliwości układu   
  z 50Mhz na częstotliwość 60 Mhz, czyli dokładnie 5 razy większą niż częstotliwość działania transmisji USB 1.1 Full Speed. Moduł układu Spartan 3E.
* **usb\_rx** – moduł odczytujący bajty i sekwencje końca / startu pakietu. Moduł został wykonany w całości przez grupę projektową.
* **sof\_filter** – moduł filtrujący ramki SOF (Start of Header), wysyłane co 1 ms   
  przez kontroler. Moduł wykonany prze grupę projektową.
* **newline\_fsm** – moduł wspomagający układ wyświetlający. Jego zadaniem było wykryć koniec pakietu i ustawić kursor w następnej linii. Moduł wykonany przez grupę projektową.
* **FSM\_SendByte** – moduł przetwarzający bajty na ich reprezentację szesnastkową   
  na ekranie. Moduł wykonany przez dr inż. Jarosława Sugiera.
* **VGAtxt48x20** – sterownik ekranu podłączonego do układu FPGA. Umożliwiał   
  on wypisywanie 48 znaków w 20 liniach na ekranie oraz przewijanie tekstu. Moduł wykonany przez dr inż. Jarosława Sugiera.

Układ nie posiada modułu **usb\_tx**, który odpowiada za transmisję danych do kontrolera, z uwagi   
na wspomnianą małą ilość czasu, jednak jego zalążki można znaleźć w plikach projektu. Moduły wykonane przez dr inż. Jarosława Sugiera znajdują się na stronie kursu [3].

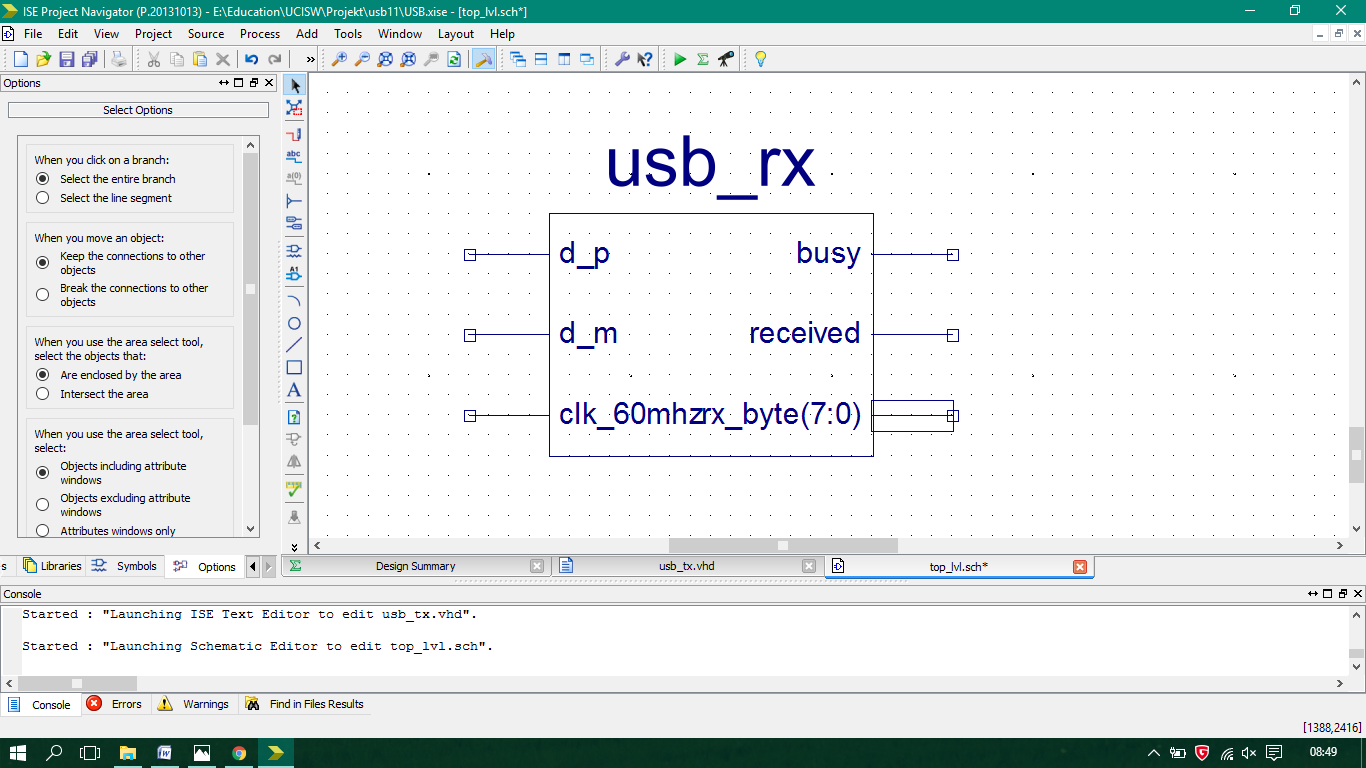


Rysunek 9. Schemat układu odbierającego pakiety z interfejsu USB 1.1 Full Speed

## 2.1. Opisy modułów

Poniżej zostały opisane moduły wykonane przez grupę:

* + 1. **usb\_rx**



Rysunek 10. Symbol układu usb\_rx.

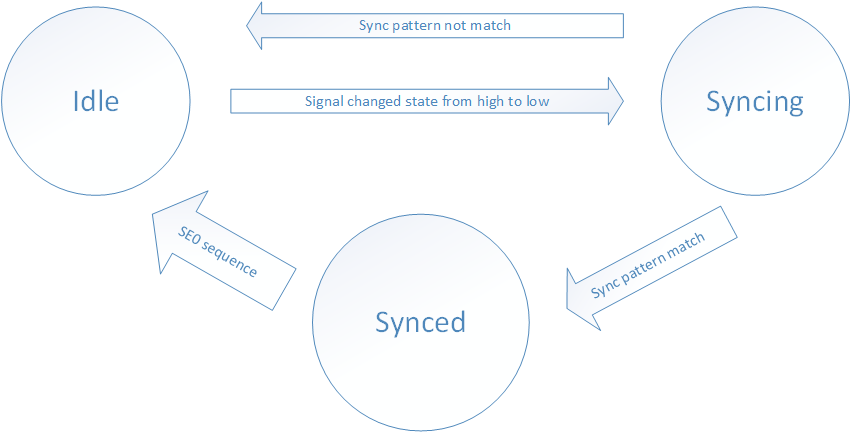
**Wejścia i wyjścia układu:**

* + d\_p i d\_m – są to bezpośrednie styki z przewodami D+ i D- interfejsu USB. W danym układzie są to wejścia.
  + clk\_60mhz – wejście zegara o częstotliwości 60 MHz.
  + busy – wyjście układu informujące, że urządzenie odbiera dane.
  + received – wyjście sygnalizujące gotowość bajtu do odbioru za pomocą 1-taktowego impulsu.
  + rx\_byte – magistrala z odebranymi danymi.

**Maszyna stanów:**

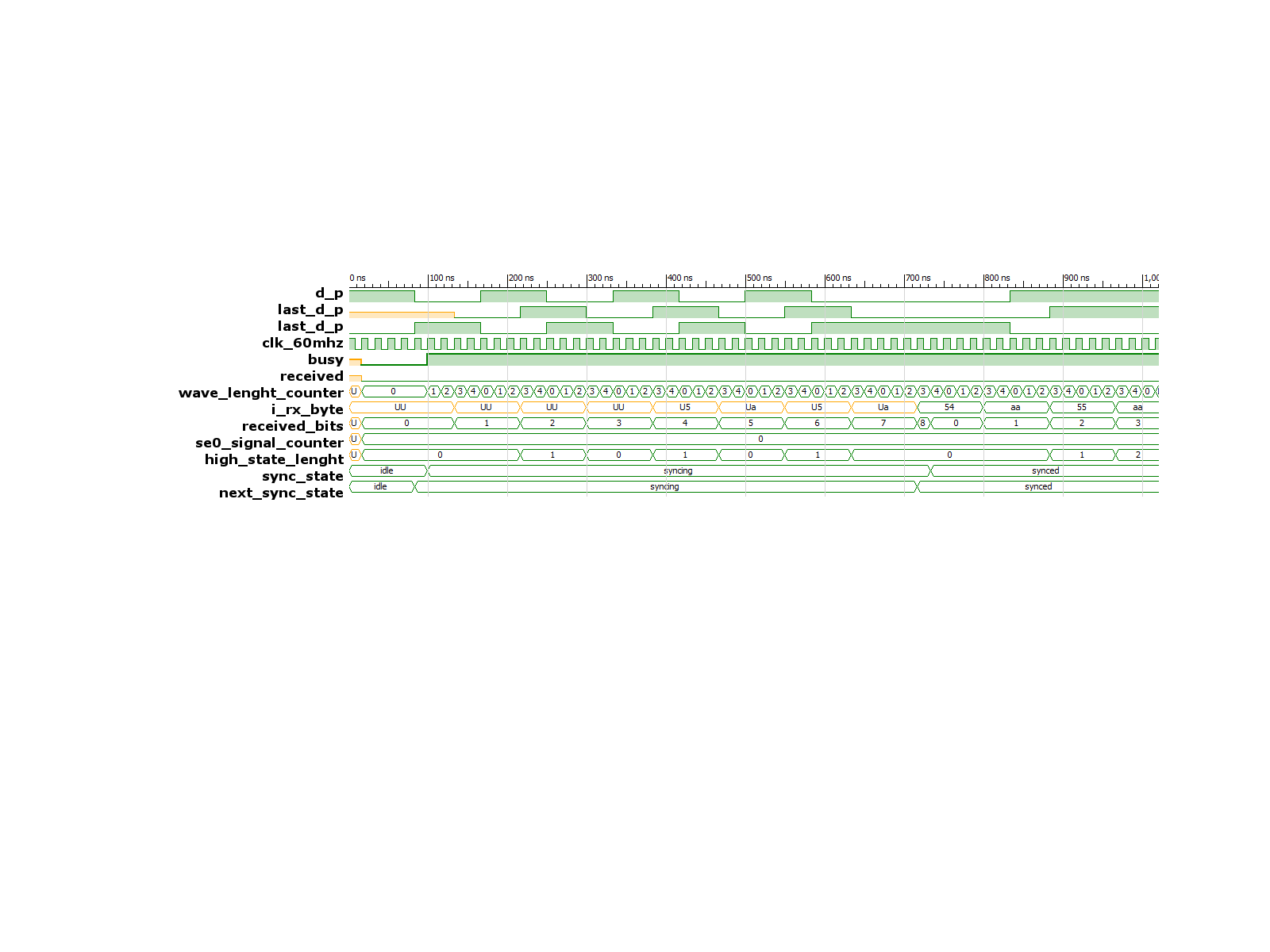
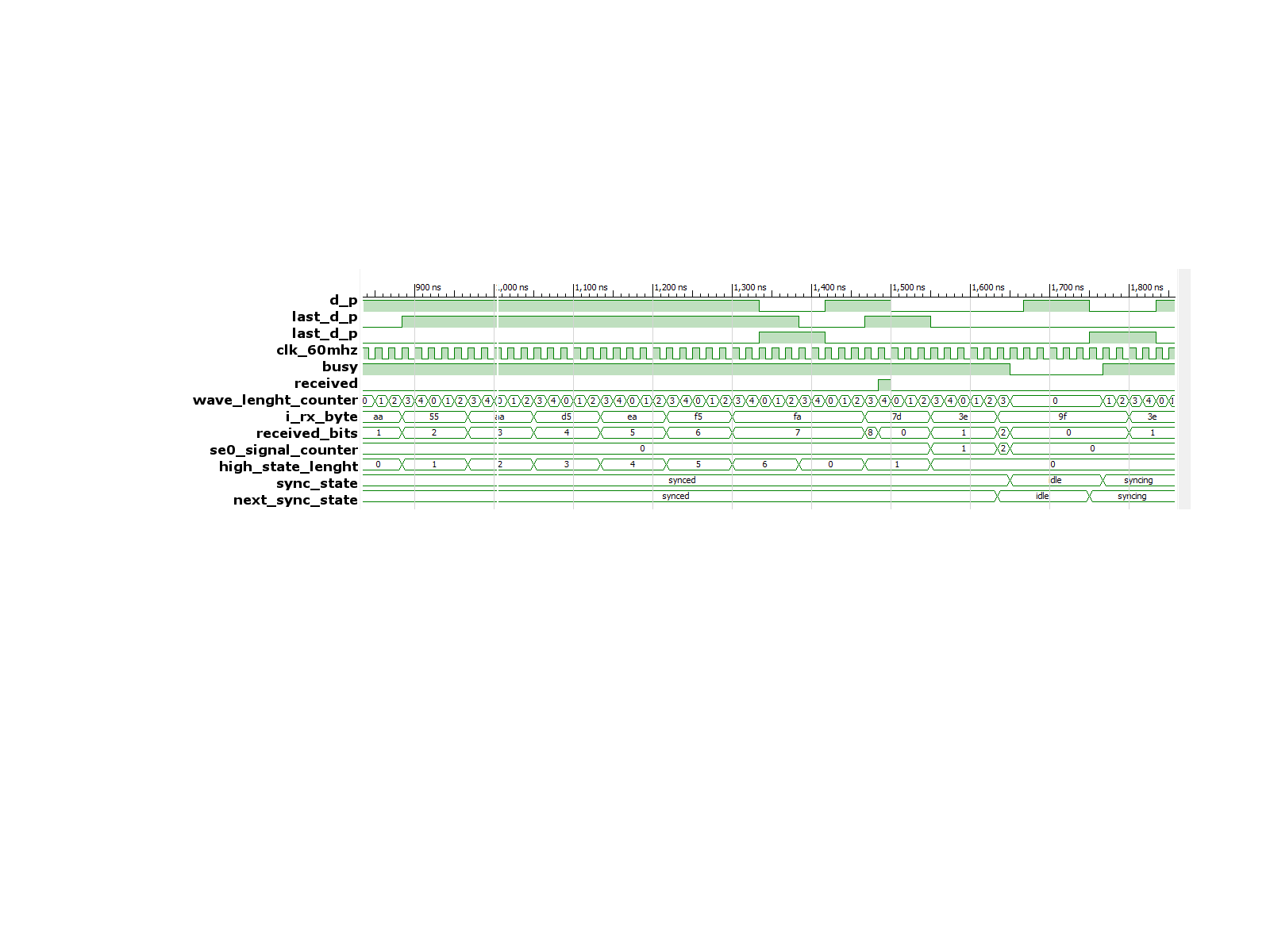
Maszyna stanów jest głównym elementem modułu usb\_rx (Rysunek 11) definiującym stan odbiornika:

* Idle – urządzenie oczekuje na rozpoczęcie sygnału synchronizacji
* Syncing – urządzenie jest w trakcie synchronizacji
* Synced – synchronizacja była udana, odbieranie bajtów z danymi



Rysunek 11. Maszyna stanów dla modułu usb\_rx.

**Symulacja**



**1)**

**5)**

**4)**

**3)**

**2)**

Rysunek 13. Symulacja odbioru i końca transmisji pakietu.

Rysunek 12. Symulacja synchronizacji transmisji pakietu.

Na rysunku 12 i 13 pokazane są przebiegi symulacji dla kolejno końca odbioru pakietu   
jak i początku. Test testuje także różne przypadki jak na przykład wielokrotne niepowodzenie synchronizowania, zbyt krótki sygnał se0, zerwanie połączenia itd.

Cyframi zostały oznaczone charakterystyczne punkty w działaniu programu:

1. Zmiana stanu sygnału na niski dla urządzenia w stanie Idle – rozpoczęcie synchronizacji
2. Zebranie 8 bitów pasujących do znaku synchronizacji – zmiana stanu na Synchronized   
   i rozpoczęcie odbioru pakietu.
3. Co kolejne 8 bitów pakietu sygnalizowany jest odbiór bajtu za pomocą jednotaktowego impulsu
4. Sygnał SE0, czyli obie linie mają stan niski – sygnał końca pakietu.
5. SE0 trwa określony czas, co powoduje przejście urządzenia w stan Idle – koniec pakietu.

Opis sygnałów (wewnętrznych):

* last\_dp – zapisuje poprzedni stan d\_p to dekodowania sygnału.
* wave\_lenght\_counter – licznik modulo 5 liczący czas do kolejnej zmiany sygnału w standardzie USB (5 razy mniejsza).
* received\_bits – zlicza liczbę odebranych bitów, przy 8 zmienia stan modułu lub sygnalizuje gotowość odbioru bajtu
* se0\_signal\_counter – zlicza czas przebywania w stanie single edge 0 (obie linie mają niski stan)
* high\_state\_lenght – czas przebywania linii w stanie wysokim, potrzebne do usuwania dodatkowych bitów
* sync\_state, next\_sync\_state – sygnały oznaczające aktualny i kolejny stan maszyny stanów

**Fragmenty programu**

Zgodnie ze specyfikacją przed rozpoczęciem przesyłania pakietu znajduje się przez dłuższy czas stan wysoki. W tym stanie urządzenie czeka na zmianę stanu linii z wysokiego na niski (operujemy głównie na linii d\_p). Gdy zmiana nadejdzie, rozpoczynana jest synchronizacja, czyli odebranie   
8 bitów. Odebranie poprawnej sekwencji zmienia stan maszyny w Synced, natomiast nieudana powraca ją do stanu czuwania.

Listing 1. Odbieranie danych z interfejsu USB.

if (rising\_edge(clk\_60mhz)) then

if(next\_sync\_state = idle or received\_bits = 8) then

received\_bits <= (others => '0');

elsif(wave\_length\_counter = 2 and high\_state\_lenght < 6) then

if(sync\_state = synced) then

i\_rx\_byte <= (last\_d\_p xnor d\_p) & i\_rx\_byte(7 downto 1);

else

i\_rx\_byte <= i\_rx\_byte(6 downto 0) & d\_p;

end if;

received\_bits <= received\_bits + 1; -- increments counter

end if;

end if;

Listing 2. Wyznaczanie pozycji wewnątrz cyklu USB.

if rising\_edge(clk\_60mhz) then

if(sync\_state = idle and next\_sync\_state = syncing) then

wave\_length\_counter <= "001";

elsif(wave\_length\_counter = 4 or next\_sync\_state = idle) then

wave\_length\_counter <= (others => '0');

else

wave\_length\_counter <= wave\_length\_counter + 1;

end if;

end if;

Listing 1 zajmuje się pobieraniem danych z interfejsu. W zależności od stanu maszyny wykonuje   
to w inny sposób. Dla stanu synchronizacji pobiera je bez zmian do porównania po zebraniu 8 bitów, w stanie zsynchronizowania dekoduje i umieszcza w tym samym rejestrze rx\_byte.

Do synchronizacji z taktowaniem interfejsu USB wykorzystywany jest licznik zerowany podczas rozpoczęcie synchronizacji. Częstotliwość zegara większa niż interfejsu pozwala na wykonanie dodatkowych operacji pomiędzy pobraniem nowe wartości z interfejsu oraz pozwala na pobieranie tego w środku taktu – w miejscu gdzie sygnał powinien być ustabilizowany.

Dodatkowy warunek sprawdzający długość stanu wysokiego w transmisji umożliwia usuwanie dodatkowych bitów wprowadzonych w przypadku zbyt długiego braku zmiany stanu. Bit   
jest kodowany wartością niską i wstawiany jest gdy przez 6 ostatnich taktów sygnał miał stan wysoki.

Listing 3. Wyznaczanie długości stanu wysokiego podczas transmisji

if(rising\_edge(clk\_60mhz)) then

if(next\_sync\_state = idle) then

high\_state\_lenght <= (others => '0');

elsif(wave\_length\_counter = 2) then

if(d\_p = '1' and d\_m = '0') then

high\_state\_lenght <= high\_state\_lenght + 1;

else

high\_state\_lenght <= (others => '0');

end if;

end if;

end if;

Listing 3 przedstawia sposób zliczania długości stanu wysokiego. W podobny sposób zliczany   
jest długość stanu se0. Z kolei Listing 4 przedstawia implementację maszyny stanów z Rysunku 11.

Listing 4. Implementacja maszyny stanów z Rysunku 11.

case sync\_state is

when idle =>

if(d\_p = '0' and d\_m = '1') then --waiting for zero to start sync

next\_sync\_state <= syncing;

end if;

when syncing => -- counts sync bits

if(received\_bits = 8) then

if(i\_rx\_byte = sync\_pattern) then

next\_sync\_state <= synced; --caught sync signal

else

next\_sync\_state <= idle; --failed - reset

end if;

end if;

when synced =>

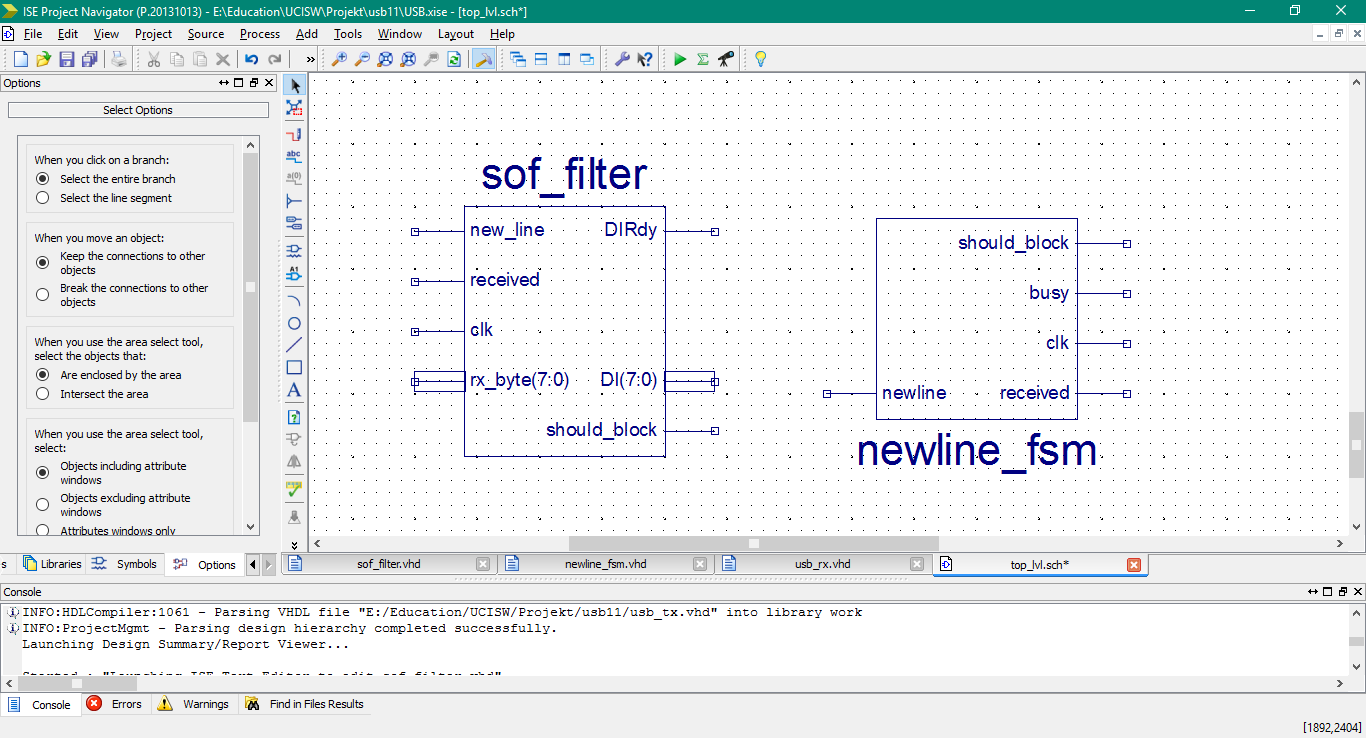
if(se0\_signal\_counter = 2) then

next\_sync\_state <= idle;

end if;

end case;

* + 1. **sof\_filter**



Rysunek 14. Symbol układu so\_filter.

Wejścia i wyjścia układu:

* DI – magistrala przepuszczająca sygnał wejściowy rx\_byte
* should\_block – sygnał wskazujący czy dany pakiet jest typu SOF (Start of Frame) i czy należy go zablokować
* DIRdy – sygnał wyzwalający wysłanie danych z DI do konwertera bitów w postać szesnastkową
* clk – zegar układu
* received – czy nowy bajt jest gotowy do odebrania
* new\_line – odbiera sygnał busy z modułu rx\_byte, który informuje o zakończeniu odbierania pakietu.

Układ filtruje pakiety SOF, które wysyłane są co 1 ms przez kontroler. Pakiety są rozpoznawalne poprzez sprawdzanie czy pierwszy bajt pakiety zaczyna się od sekwencji 0xA5 (Listing 5). Początek pakietu jest określany poprzez wykrycie impulsu sygnalizującego nową linię. Dany sposób ma wadę powodującą niefiltrowanie pierwszego pakietu SOF (Listing 6).

Listing 5. Ustalenie, czy sygnał powinien być blokowany

if(rising\_edge(clk) and first\_packet = '1') then

if(rx\_byte = X"A5") then

i\_should\_block <= '1';

else

i\_should\_block <= '0';

end if;

end if;

Listing 6. Wykrywanie początku pakietu.

if(rising\_edge(clk)) then

if(received = '1' and first\_packet = '1') then

first\_packet <= '0';

elsif(last\_newline = '1' and new\_line = '0') then

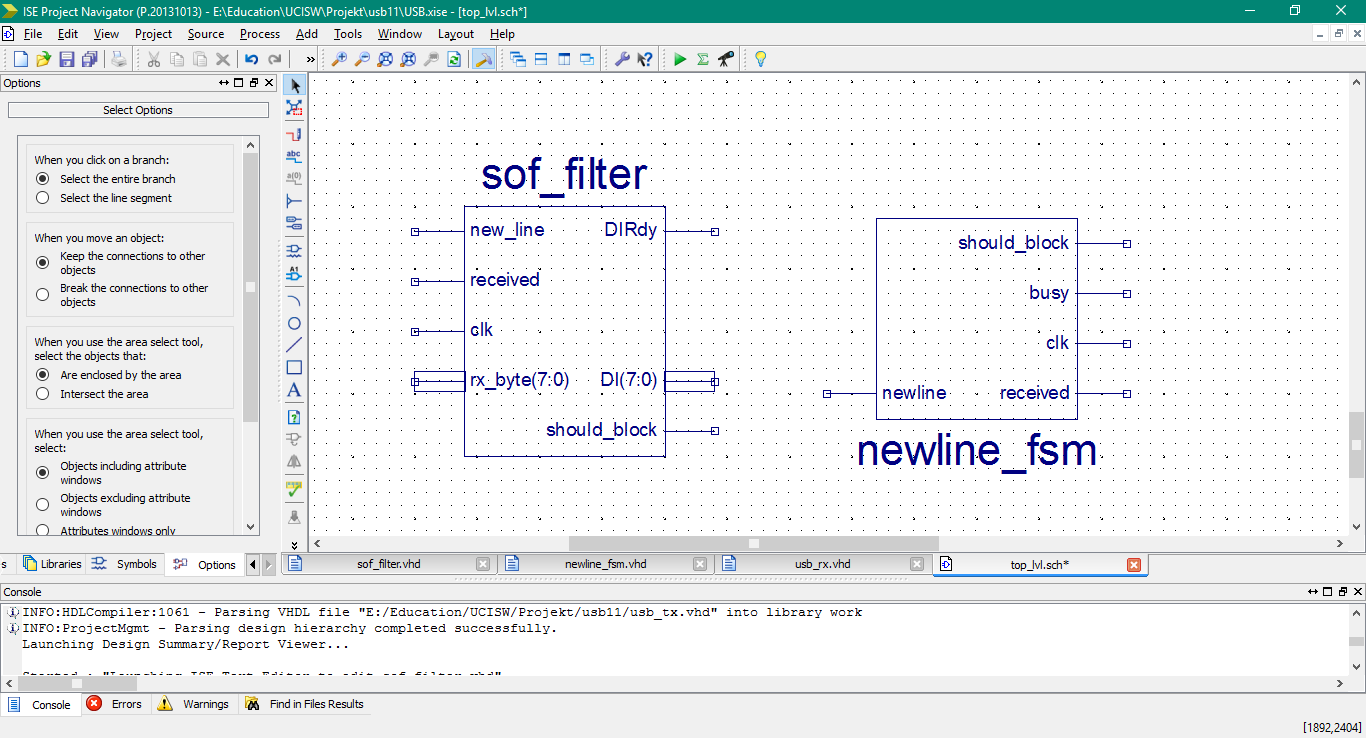
first\_packet <= '1';

end if;

last\_newline <= new\_line;

end if

* + 1. **newline\_fsm**



Rysunek 15. Symbol układu newline\_fsm.

Wejścia i wyjścia układu:

* clk – zegar układu
* received – czy nowy bajt został udostępniony
* busy – czy moduł usb\_rx jest zajęty odbieraniem
* should\_block – czy znak nowej linii powinien być zablokowany
* newline – sygnał biegnący do sterownika ekranu informujący o zmianie linii

Kolejny moduł pomocniczy do wyświetlania danych na ekranie. Układ sprawdza czy wystąpił impuls odebrania pakietu i przygotowuje się do wysłania sygnału nowej linii. Następnie sygnał   
jest wysyłany jeśli urządzenie skończyło pracę (to był ostatni pakiet) oraz nie jest on zablokowany przez moduł sof\_filter (Listing 8).

Listing 7. Ustalenie, czy powinno zostać wyświetlona nowa linia.

if(rising\_edge(clk)) then

notify <= (not i\_newline) and ((not received) and last\_received);

end if;

Listing 8. Oczekiwanie na wyświetlenie nowej linii.

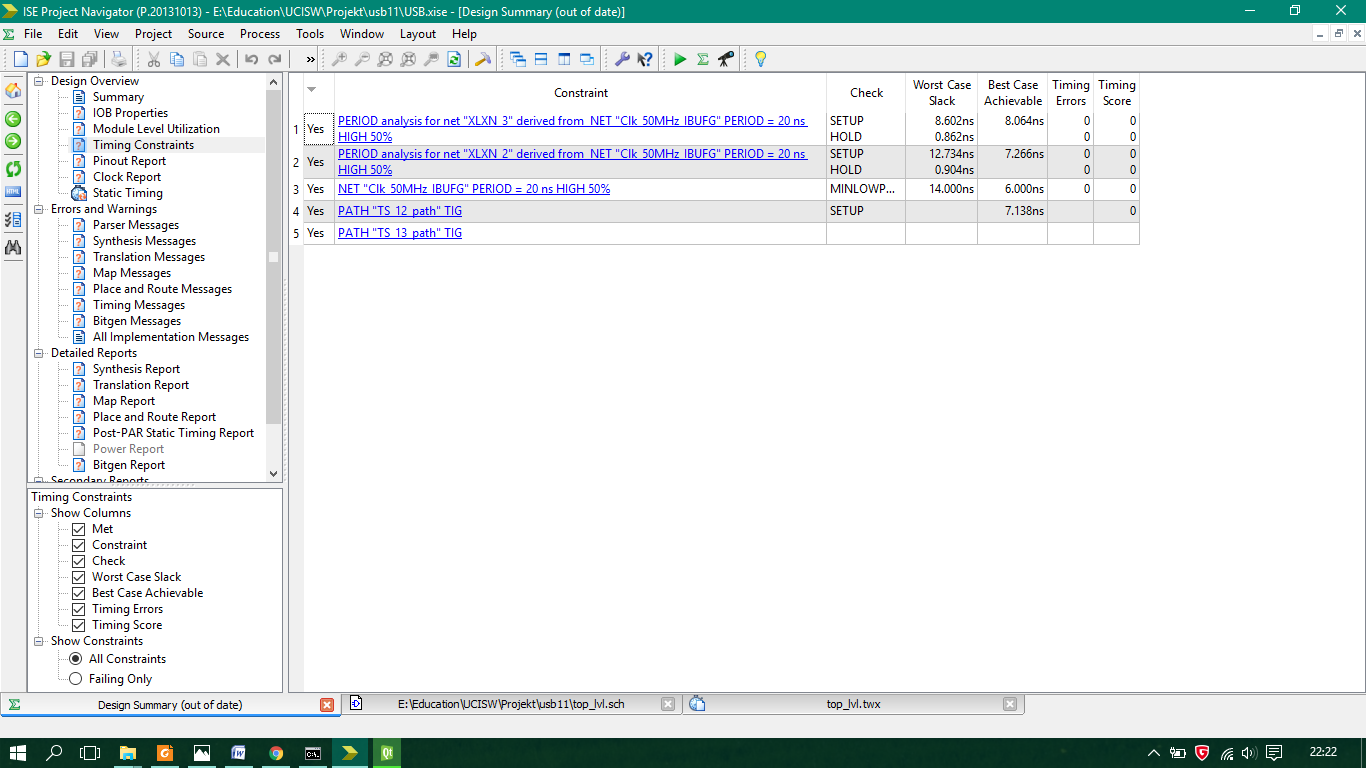
if(rising\_edge(clk)) then

i\_newline <= (not i\_newline) and ((notify and (not busy)) and (not should\_block));

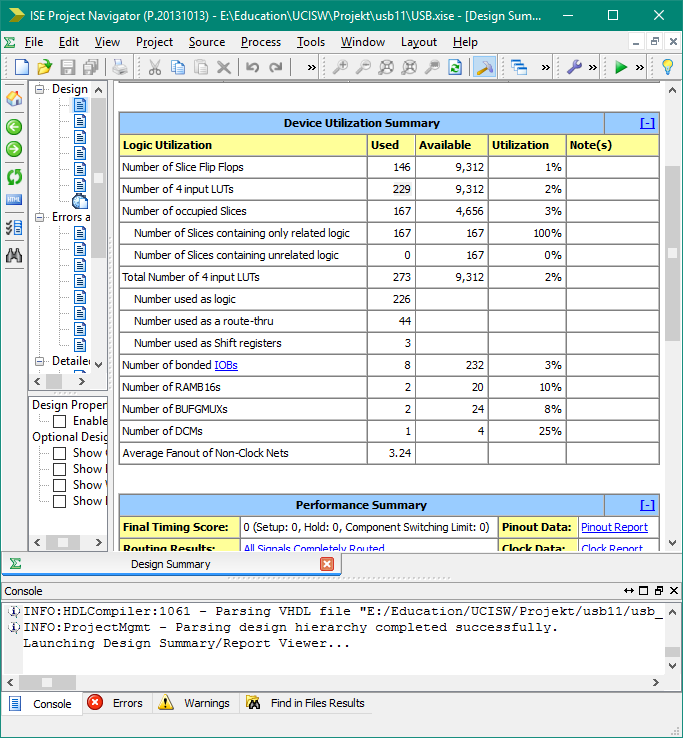
end if;

# 3. Implementacja programu w układzie FPGA

Implementacja kodu układu przebiega bez zakłóceń, a jedynymi ostrzeżeniami jaki otrzymujemy   
są niepodłączone wejścia użytych modułów. Zgodnie z raportem dotyczącym zależności czasowych   
w programie, układ zawiera rezerwę umożliwiającą prawidłowe działanie nawet przy nieznacznym zwiększeniu długości aktualnych ścieżek. Osiągnięte rezultaty przedstawia rysunek 16. Zgodnie   
z poniższym raportem stworzony układ potrafi działać z częstotliwością maksymalną **124 MHz**,   
czyli okres powinien trwać nie krócej niż **8.064ns**. Zużycie zasobów przedstawione jest w tabeli 1. Zużycie tych zasobów jest minimalne, umożliwiające stworzenie innego układu obok odbiornika USB.



Rysunek 16. Raport czasu propagacji w układzie.

Tabela 3. Zużycie zasobów.

# 4. Instrukcja użytkownika

# E:\Education\UCISW\Projekt\dokumentacja\IMG_20160419_163305779.jpg

Rysunek 17. Wynik działania programu.

Do układu FPGA należy podłączyć port USB oraz ekran do wejścia VGA. Po zaprogramowaniu układu FPGA wygenerowanym programem można podłączyć układ za pośrednictwem przewodu USB do kontrolera USB np. komputera. Po jego podłączeniu powinien pojawić się obraz ukazany   
na Rysunku 17. Zwracane dane powinny być identyczne lub bardzo podobne do tych podanych   
na Rysunku 17. Kolejne podłączanie układu FPGA do kontrolera nie zmieni obrazu ze względu   
na ciągłe wysyłanie tych samych danych przez kontroler i niezauważalnego odświeżania danych.

**4.1 Znaczenie odebranych danych**

*Ramki zaczynające się na 0xA5*

Ramki te nazywają się SOF (Start Of Frame) i są wysyłane cyklicznie co 1 ms. Kolejne bity (bez 5 ostatnich oznaczających kod CRC) są kolejnym numerem ramki. Odbiór tych ramek ułatwia debugowanie układu. W tej sekcji rozwijane są informacje zawarte w opisie teoretycznym   
(punkt 1.1).

*0x2D0010*

1. **0x2D** – prefix oznaczający ramkę zawierająca informację o inicjalizacji danego urządzenia
2. **0x001 (0b00000000000)** – adres urządzenia i jego endpoint number
3. **0x10 (0b10000)** – kod CRC 5 bitowy

*0xC38006000100004000DD94*

1. **0xC8** – prefix oznaczający wysyłanie danych (w tym przypadku związanych z inicjalizacją)
2. **0x*8006000100004000*:**
   1. 0x80 – dane od hosta dla kontrolera, standardowe zapytanie dla urządzenia
   2. 0x06 – typ żądania: GET\_DESCRIPTOR
   3. 0x0001 – typ deskryptora – DEVICE
   4. 0x0000 – same zera albo LANGUAGE\_ID
   5. 0x4000 – długość dekryptora
3. **0xDD94** – kod CRC 16 bitowy

Obie dane są danymi inicjalizującymi klienta kontrolera, dane powinny zostać akceptowane pakietem ACK do ustabilizowania połączenia.

# 5. Podsumowanie

Największą zaletą projektu jest poznanie działania interfejsu USB, który jest szeroko używany w dzisiejszym świecie oraz jest nadal rozwijany.

## 5.1. Ocena projektu

Największą wadą projektu jest brak nadajnika USB, co uniemożliwia komunikację z kontrolerem wymaganą do przesyłania danych. Kolejnym problemem jest niezaimplementowanie kontroli CRC, co powinno zostać wykonane w wyższej (warstwa protokołu) warstwie połączenia.

Zgodnie z koncepcją ustaloną w połowie zajęć projektowych należałoby wykonać 3 moduły VHDL. Jeden, odpowiadający za konwersje sygnału w paczki danych został wykonany. Kolejne dwa odpowiadałyby za protokół USB oraz umożliwiałyby przesyłanie danych bez znajomości protokołu.

Testowanie różnych przypadków odbieranego sygnału – w ramach tolerancji uwzględnionej   
w specyfikacji przebiegło pomyślnie, co daje nadzieję, że moduł został wykonany dość dobrze.

Wyświetlanie odbieranych bajtów zostało wykonane bez należytej staranności jedynie w celach podglądowych – wykonanie lepiej tego mechanizmu i zamknięcie go w 1 module z pewnością przydałoby się w dalszych etapach prac.

## 5.2. Dalsza praca

Jak zostało wspomniane, implementacja pozostałych dwóch warstw połączenia znacznie ułatwi pracę z odbieranymi bajtami. Nadajnik USB (także wykonany warstwowo) jest niezbędny dla dalszego postępu z projektem. Kontrola CRC znacznie zwiększy niezawodność przesyłania danych.

Oprócz powyższych, bardziej hardwarowych prac należałoby odpowiadać prawidłowo   
na odpowiednie zapytania kontrolera, dzięki czemu urządzenie USB byłoby rozpoznawalne   
jako prawidłowe.

Analiza złożoności układu (Tabela 3) wskazuje na możliwość wprowadzenia jeszcze wielu funkcjonalności w budowanym module USB.

# 6. Bibliografia

1. Dokumentacja standardu USB [Dostęp 13.05.2016]  
   http://www.usb.org/developers/docs/
2. Kopia dokumentacji USB 1.1, Universal Serial Bus Specification [Dostęp 13.05.2016]  
   <http://esd.cs.ucr.edu/webres/usb11.pdf>
3. Strona zajęć projektowych kursu Urządzenia Cyfrowa i Systemy wbudowane  
   [Dostęp 13.05.2016]  
   <http://www.zsk.ict.pwr.wroc.pl/zsk_ftp/fpga/>
4. Instrukcja układu FPGA Spartan 3e starter kit [Dostęp 13.05.2016]  
   <http://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf>
5. Opis układu XC3S500E [Dostęp 13.05.2016]  
   <http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf>
6. Schemat złącz USB i tabela przewodów [Dostęp 13.05.2016]  
   <http://forum.videohelp.com/images/guides/p1826452/usbcable.jpg>