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

1. Wstęp

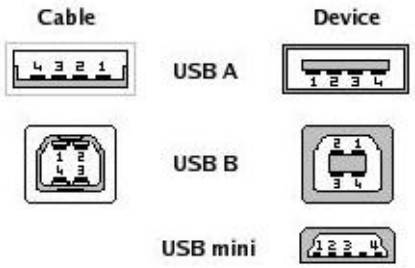
Celem projektu było zaprogramowanie w języku VHDL sprzętowej implementacji sterownika USB standardzie 1.1 Full Speed, za pomocą którego można przesyłać dane. W tym celu otrzymaliśmy możliwość korzystania z komputera z zainstalowanym oprogramowaniem ISE Webpack do tworzenia programu oraz układ FPGA Spartan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

1.1 Opis teoretyczny

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



Pin	Signal	Color	Description
1	VCC		+5V
2	D-		Data -
3	D+		Data +
4	GND		Ground

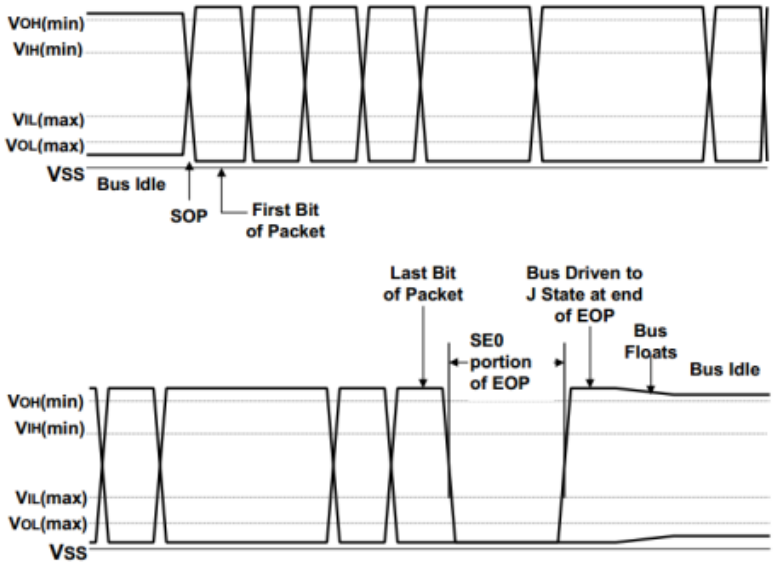
Rysunek 1. Budowa złącz interfejsu USB wraz z opisem pinów [6].

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.

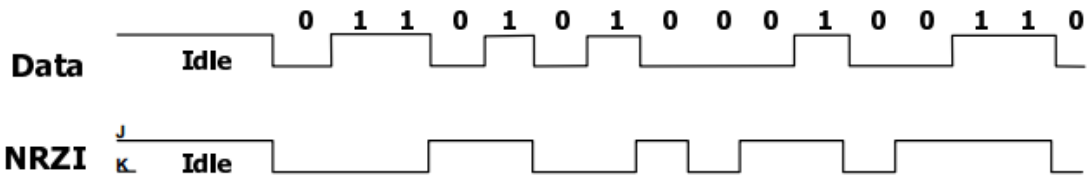


Rysunek 2. Początek i koniec pakietu.

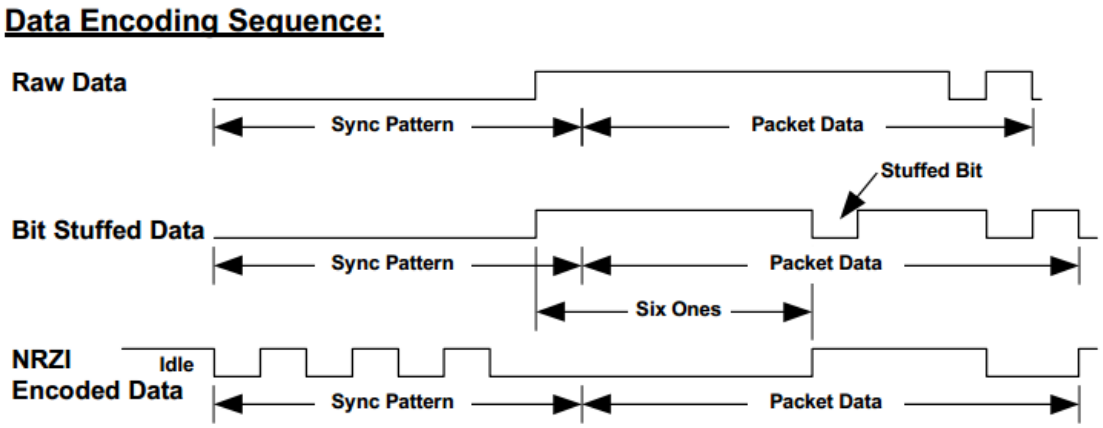
Tabela 1. Określenie sygnałów w interfejsie USB 1.1

Bus State	Signaling Levels		
	At originating source connector (at end of bit time)	At final target connector	
		Required	Acceptable
Differential "1"	D+ > V_{OH} (min) and D- < V_{OL} (max)	(D+) - (D-) > 200mV and D+ > V_{IH} (min)	(D+) - (D-) > 200mV
Differential "0"	D- > V_{OH} (min) and D+ < V_{OL} (max)	(D-) - (D+) > 200mV and D- > V_{IH} (min)	(D-) - (D+) > 200mV
Single-ended 0 (SE0)	D+ and D- < V_{OL} (max)	D+ and D- < V_{IL} (max)	D+ and D- < V_{IH} (min)
Data J state: Low-speed Full-speed	 Differential "0" Differential "1"	 Differential "0" Differential "1"	
Data K state: Low-speed Full-speed	 Differential "1" Differential "0"	 Differential "1" Differential "0"	
Idle state: Low-speed Full-speed	N.A.	 D- > V_{IHZ} (min) and D+ < V_{IL} (max) D+ > V_{IHZ} (min) and D- < V_{IL} (max)	 D- > V_{IHZ} (min) and D+ < V_{IH} (min) D+ > V_{IHZ} (min) and D- < V_{IH} (min)
Resume state	Data K state	Data K state	
Start-of-Packet (SOP)	Data lines switch from Idle to K state		
End-of-Packet (EOP)[4]	SE0 for approximately 2 bit times[1] followed by a J for 1 bit time[3]	SE0 for ≥ 1 bit time[2] followed by a J state for 1 bit time	SE0 for ≥ 1 bit time[2] followed by a J state
Disconnect (at downstream port)	N.A.	SE0 for ≥2.5μs	
Connect (at downstream port)	N.A.	Idle for ≥2ms	Idle for ≥2.5 μs
Reset	D+ and D- < V_{OL} (max) for ≥10ms	D+ and D- < V_{IL} (max) for ≥10ms	D+ and D- < V_{IL} (max) for ≥2.5 μs

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.



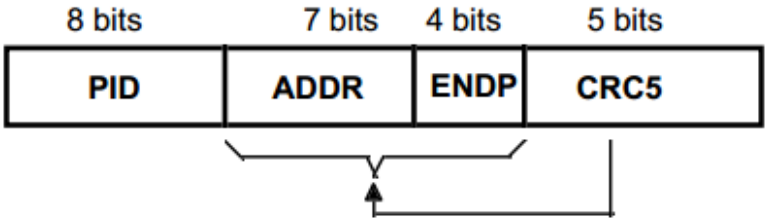
Rysunek 4. Przykład bit stuffing'u oraz sync pattern'u.

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:

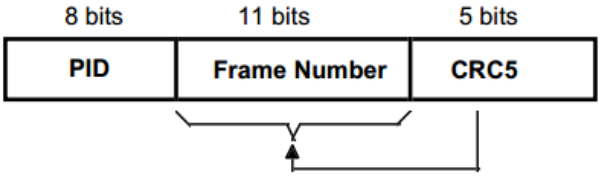
- SETUP – pakiet zawierający informacje do konfiguracji urządzenia (Rysunek 5)
- SOF – Start Of Frame, pakiet wysyłany co 1 ms przez kontroler (Rysunek 6)
- DATA – pakiet z danymi (Rysunek 7)

Tabela 2. Typy PID.

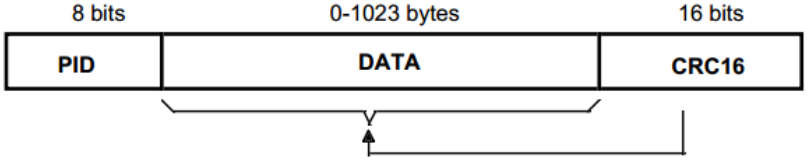
PID Type	PID Name	PID[3:0]*	Description
Token	OUT	0001B	Address + endpoint number in host-to-function transaction
	IN	1001B	Address + endpoint number in function-to-host transaction
	SOF	0101B	Start-of-Frame marker and frame number
	SETUP	1101B	Address + endpoint number in host-to-function transaction for SETUP to a control pipe
Data	DATA0	0011B	Data packet PID even
	DATA1	1011B	Data packet PID odd
Handshake	ACK	0010B	Receiver accepts error-free data packet
	NAK	1010B	Rx device cannot accept data or Tx device cannot send data
	STALL	1110B	Endpoint is halted or a control pipe request is not supported.
Special	PRE	1100B	Host-issued preamble. Enables downstream bus traffic to low-speed devices.



Rysunek 5. Pakiet typu Token (np. SETUP)

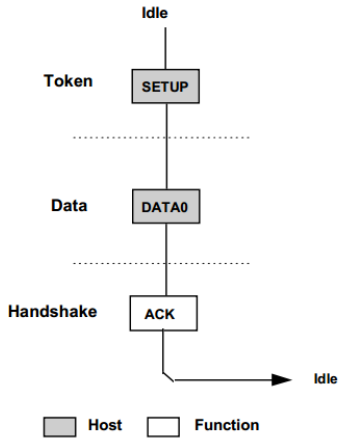


Rysunek 6. Pakiet Start Of Frame



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.



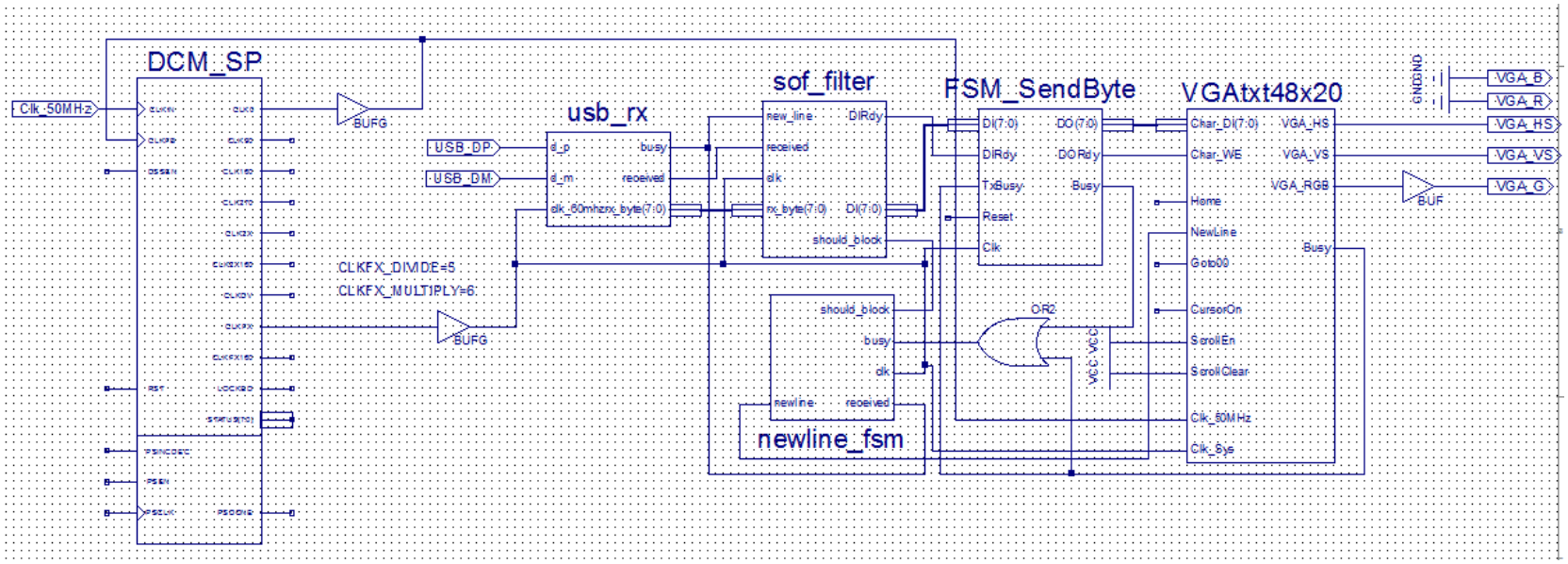
Rysunek 8. Schemat komunikacji podczas podłączania urządzenia.

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].

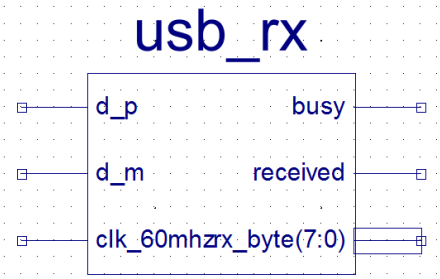


Rysunek 9. Schemat układu odbierającego pakiety z interfejsu USB 1.1 Full Speed

2.1. Opisy modułów

Poniżej zostały opisane moduły wykonane przez grupę:

2.1.1 usb_rx



Rysunek 10. Symbol układu usb_rx.

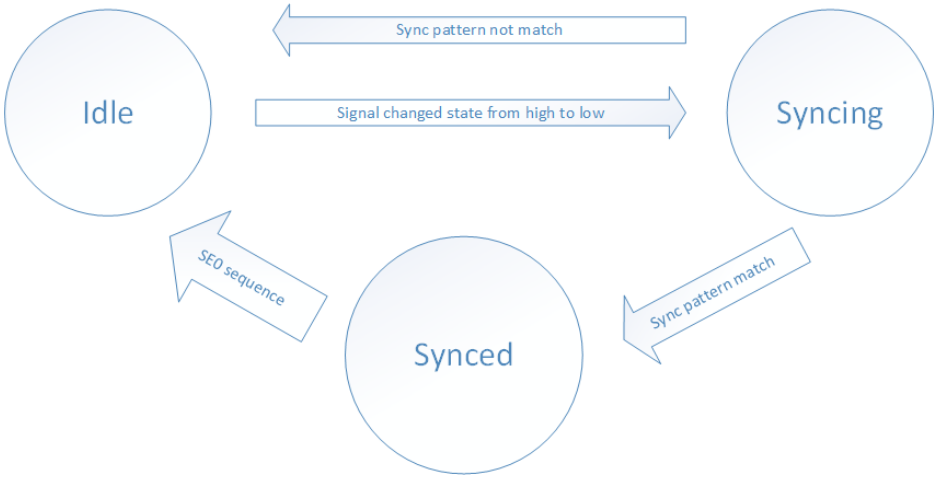
Wejścia i wyjścia układu:

- 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:

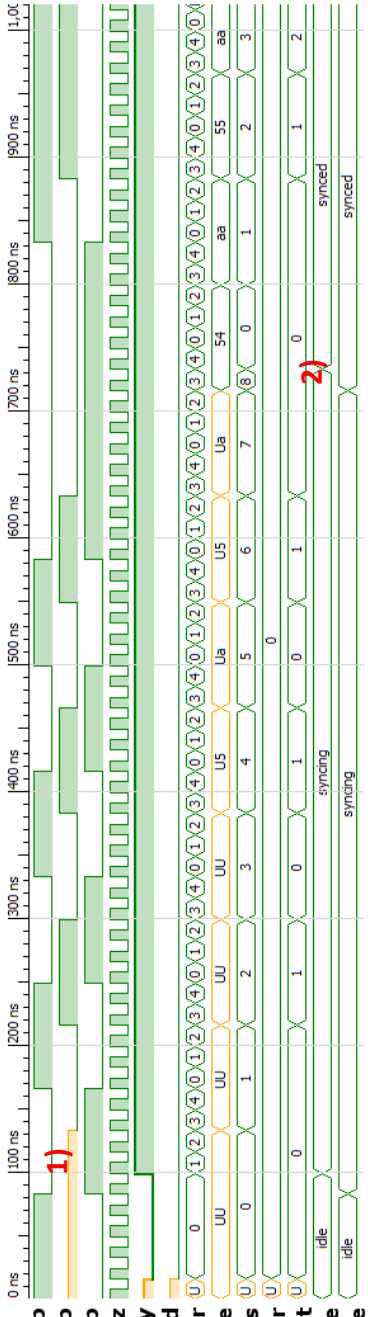
Maszyna stanów jest głównym elementem modułu usb_rx (Rysunek 11) definiującym stan odbiornika:

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

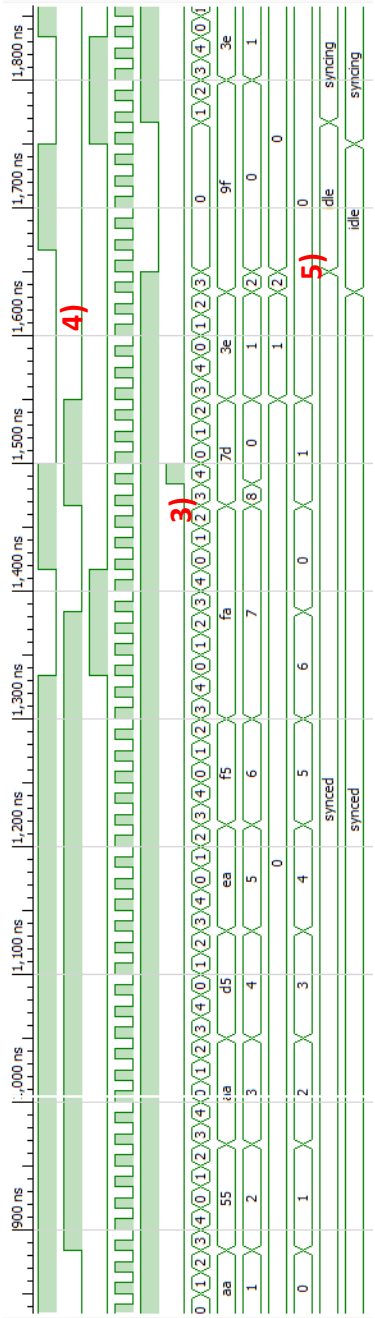


Rysunek 11. Maszyna stanów dla modułu usb_rx.

Symulacja



Rysunek 12. Symulacja synchronizacji transmisji pakietu.



Rysunek 13. Symulacja odbioru i końca 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

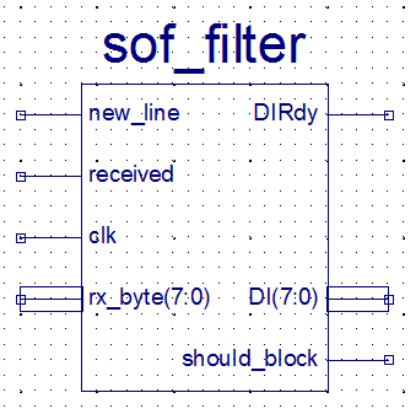
```
if(rising_edge(clk_60mhz)) then
        if(next_sync_state = idle) then
                high_state_lenght <= (others => '0');
        elsif(wave_length_counter = 2) then
                if(d_p = '1' and d_m = '0') then
                        high_state_lenght <= high_state_lenght + 1;
                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;
```

2.1.2 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).

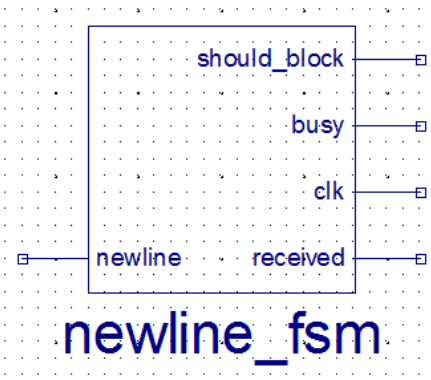
Listing 5. Ustalenie, czy sygnał powinien być blokowany

```
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
```

2.1.3 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.

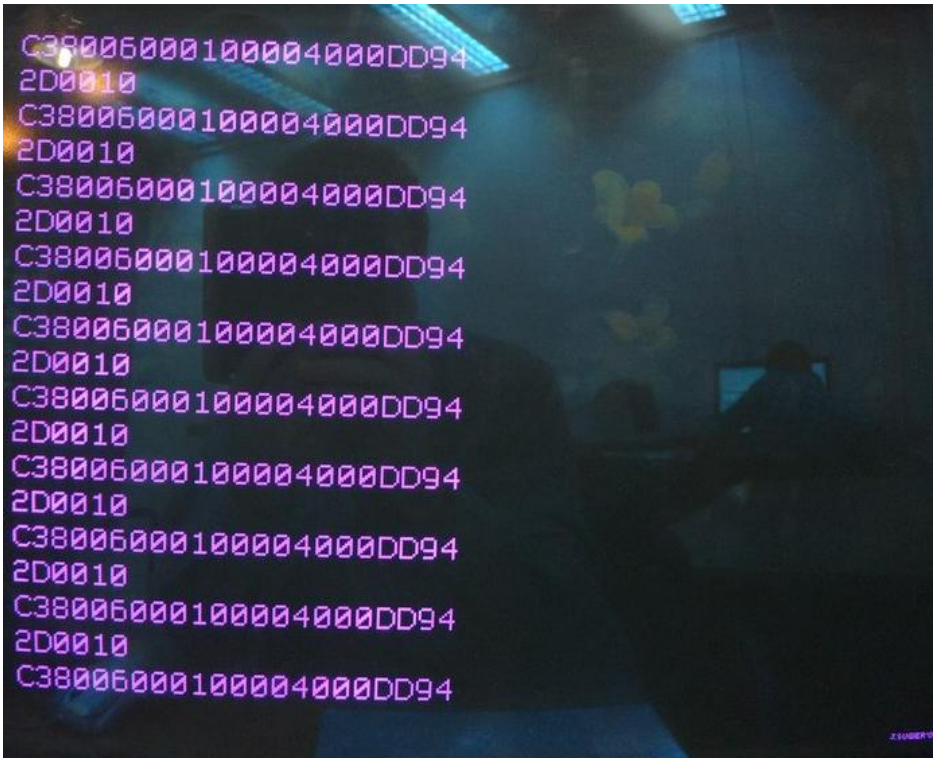
Constraint	Check	Worst Case Slack	Best Case Achievable
PERIOD analysis for net "XLXN_3" derived from NET "Clk_50MHz_IBUFG" PERIOD = 20 ns HIGH 50%	SETUP HOLD	8.602ns 0.862ns	8.064ns
PERIOD analysis for net "XLXN_2" derived from NET "Clk_50MHz_IBUFG" PERIOD = 20 ns HIGH 50%	SETUP HOLD	12.734ns 0.904ns	7.266ns
NET "Clk_50MHz_IBUFG" PERIOD = 20 ns HIGH 50%	MINLOWP...	14.000ns	6.000ns
PATH "TS_12_path" TIG	SETUP		7.138ns
PATH "TS_13_path" TIG			

Rysunek 16. Raport czasu propagacji w układzie.

Tabela 3. Zużycie zasobów.

Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	146	9,312	1%
Number of 4 input LUTs	229	9,312	2%
Number of occupied Slices	167	4,656	3%
Number of Slices containing only related logic	167	167	100%
Number of Slices containing unrelated logic	0	167	0%
Total Number of 4 input LUTs	273	9,312	2%
Number used as logic	226		
Number used as a route-thru	44		
Number used as Shift registers	3		
Number of bonded IOBs	8	232	3%
Number of RAMB16s	2	20	10%
Number of BUFGMUXs	2	24	8%
Number of DCMs	1	4	25%
Average Fanout of Non-Clock Nets	3.24		

4. Instrukcja użytkownika



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

3 **0x2D** – prefix oznaczający ramkę zawierająca informację o inicjalizacji danego urządzenia
4 **0x001 (0b00000000000)** – adres urządzenia i jego endpoint number
5 **0x10 (0b10000)** – kod CRC 5 bitowy

0xC38006000100004000DD94

6 **0xC8** – prefix oznaczający wysyłanie danych (w tym przypadku związanych z inicjalizacją)
7 **0x*8006000100004000*:**
7.1 0x80 – dane od hosta dla kontrolera, standardowe zapytanie dla urządzenia
7.2 0x06 – typ żądania: GET_DESCRIPTOR
7.3 0x0001 – typ deskryptora – DEVICE
7.4 0x0000 – same zera albo LANGUAGE_ID
7.5 0x4000 – długość dekryptora
8 **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