AGH
Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
Wydział Fizyki i Informatyki Stosowanej

Praca inżynierska

Jakub Moroń

Mirosław Firlej

kierunek studiów: **fizyka techniczna**

kierunek dyplomowania: **fizyka jądrowa**

Projektowanie interfejsu USB w oparciu o programowalne układy logiczne (Field Programmable Gate Array)

Opiekun: **dr hab. inż. Marek Idzik**

Kraków, luty 2009

Oświadczamy, świadomi odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonaliśmy osobiście i samodzielnie i nie korzystaliśmy ze źródeł innych niż wymienione w pracy.

Merytoryczna ocena pracy Jakuba Moronia przez opiekuna:

Końcowa ocena pracy przez opiekuna: .....

Data: ..... Podpis: .....

Merytoryczna ocena pracy Jakuba Moronia przez recenzenta:

Końcowa ocena pracy przez recenzenta: .....

Data: ..... Podpis: .....

Skala ocen: (6.0 – celująca), 5.0 – bardzo dobra, 4.5 – plus dobra, 4.0 – dobra, 3.5 – plus dostateczna, 3.0 – dostatetczna, 2.0 – niedostatetczna

Merytoryczna ocena pracy Mirosława Firleja przez opiekuna:

Końcowa ocena pracy przez opiekuna: .....

Data: ..... Podpis: .....

Merytoryczna ocena pracy Mirosława Firleja przez recenzenta:

Końcowa ocena pracy przez recenzenta: .....

Data: ..... Podpis: .....

Skala ocen: (6.0 – celująca), 5.0 – bardzo dobra, 4.5 – plus dobra, 4.0 – dobra, 3.5 – plus dostateczna, 3.0 – dostatetczna, 2.0 – niedostatetczna

Spis treści

1 Wstęp

1.1 Cel i zakres pracy

Celem pracy, który został postawiony do realizacji, było zaprojektowanie i wykonanie dwukierunkowego interfejsu komunikacyjnego pracującego na magistrali USB. Jako sposób fizycznej realizacji układu wybrano syntezę logiczną wewnątrz programowalnego układu logicznego (FPGA). Interfejs USB ma być wykorzystywany podczas testowania prototypowych układów scalonych pracujących przy wysokich częstotliwościach, projektowanych w Zakładzie Elektroniki Jądrowej i Detekcji Promieniowania, Katedry Oddziaływań i Detekcji Cząstek na Wydziale Fizyki i Informatyki Stosowanej AGH.

Do celów zbierania danych, ze względu na prostotę obsługi i oprogramowania, często stosowany jest wychodzący z użycia interfejs RS-232. Niska przepustowość tego portu sprawia, że czas przesyłania danych z systemu pomiarowego (systemu akwizycji danych) do komputera jest bardzo długi. Dodatkową wadą jest słaba dostępność portu RS-232 na nowych płytach głównych, nie mówiąc o komputerach przenośnych, które go nie posiadają. Aby usprawnić proces zbierania (akwizycji) danych, konieczne stało się wykorzystanie dobrego i popularnego standardu, jakim jest USB.

Do fizycznego wykonania interfejsu USB wykorzystano układ FPGA, ze względu na szybkość działania i możliwość wielokrotnej rekonfiguracji. Zdecydowano się na interfejs pracujący w standardzie Full-Speed, ponieważ nie wymaga on bardzo szybkiego układu Transceiver'a. Dzięki temu cały interfejs może być układem logicznym zaimplementowanym wewnątrz układu programowalnego. Dodatkowym założeniem projektowym była uniwersalność projektu, czyli możliwość implementacji interfejsu na dowolnym układzie FPGA. Gotowy projekt powinien stanowić oddzielny moduł (rdzeń), który może być użyty jako interfejs komunikacyjny (bez wchodzenia w strukturę wewnętrzną) w dowolnym układzie pomiarowym.

1.2 Streszczenie

W rozdziale 2 zawarto krótki opis zagadnień teoretycznych wykorzystywanych w opisie uniwersalnej magistrali szeregowej (USB). Podstawowe pojęcia. Parametry sygnałów przesyłanych po USB. Bity synchronizacyjne. Kodowanie NRZI. Skrócony opis protokołu komunikacyjnego.

W rozdziale 3 przedstawiono środowisko programistyczne i jego możliwości. Krótki opis płytki testowej i ważniejszych elementów wchodzących w jej skład.

W rozdziale 4 zamieszczono ogólny opis Transceiver'a USB. Przedstawiono sposób nadawania i odbierania pakietów. Moment przełączenia interfejsu z trybu odbioru, w tryb nadawania.

W rozdziale 5 przedstawiono szczegółowy opis wszystkich bloków funkcjonalnych układu Transceiver'a. Przebiegi logiczne. Dokładne omówienie każdego z sygnałów.

W rozdziale 6 zawarto ogólny opis działania logiki stosu uniwersalnej magistrali szeregowej (USB). Współpraca z układem Transceiver'a. Podłączenie bloków miedzy sobą.

W rozdziale 7 zamieszczono szczegółowy opis wszystkich bloków funkcjonalnych logiki stosu USB. Opis działania maszyny stanów skończonych - głównego układu zarządzającego pracą stosu.

W rozdziale 8 zawarto opis układu wykonawczego. Schemat i projekt płytki drukowanej. Konieczność zastosowania zewnętrznego układu buforującego i specjalnego generatora przebiegu zegarowego.

W rozdziale 9 przedstawiono sposoby testowania poprawności działania interfejsu USB. Analizator stanów logicznych zbudowany na mikrokontrolerze ATMEGA162 oraz program do obliczania cyklicznego kodu nadmiarowego (CRC). Opis działania układu interfejsu USB pod systemem operacyjnym *Linux*.

W rozdziale 10 zamieszczono podsumowanie i wnioski. Poziom realizacji założonych zadań. Zakres wiedzy i umiejętności wykorzystany podczas realizacji tematu.

W rozdziale 11 zawarto zbiór dokumentacji pomocny podczas projektowania interfejsu USB.

1.3 Podział Pracy

W ramach niniejszej pracy przyjęto następujący podział zadań:

Mirosław Firlej zaprojektował:

- część sprzętową interfejsu USB (Transceiver), opisaną w rozdziałach 4 i 5.
- zewnętrzny układ wykonawczy opisany w rozdziale 8.
- program do obliczania cyklicznego kodu nadmiarowego CRC (podrozdział 9.1).
- 24-bitowy analizator stanów logicznych (podrozdział 9.2).

Jakub Moroń zaprojektował:

- część logiczną interfejsu USB (Stos), opisaną w rozdziałach 6 i 7

Wspólnie wykonano:

- opis zagadnień teoretycznych (rozdział 2).
- przedstawienie środowiska programistycznego i płytki testowej (rozdział 3).
- podsumowanie (rozdział 10).

2 Wybrane zagadnienia teoretyczne dotyczące USB

USB (Universal Serial Bus – Uniwersalna Magistrala Szeregowa) jest rodzajem interfejsu komunikacyjnego zastępującego w komputerach starsze porty szeregowe i równoległe. Jej podstawową zaletą jest prostota obsługi (gotowego urządzenia) pod wieloma systemami operacyjnymi oraz możliwość podłączenia do 127 urządzeń do jednego kontrolera. Sama budowa urządzenia pracującego na USB nie jest jednak prosta, ponieważ wymaga od projektanta spełnienia szeregu zasad i standardów wyszczególnionych w specyfikacji USB[1]. W pracy ograniczono się jedynie do przedstawienia najważniejszych zagadnień teoretycznych związanych z działaniem samej magistrali, a także protokołu komunikacyjnego używanego w projekcie interfejsu USB.

2.1 Typy i prędkości USB

Urządzenia pracujące na magistrali szeregowej można podzielić na grupy w zależności od wersji standardu:

- **USB 1.1** - Urządzenia pracujące z prędkościami $12Mb/s$ (Full-Speed) lub $1.5Mb/s$ (Low-Speed)
- **USB 2.0** - Urządzenia pracujące z prędkością $480Mb/s$ (High-Speed) i kompatybilne ze standardami wcześniejszymi USB 1.1
- **USB 3.0** - Urządzenia pracujące z prędkością $4,8Gb/s$ (Super-Speed) i kompatybilne ze standardami USB 1.1 i USB 2.0

2.2 Ważne terminy i pojęcia

Zawarte w tym rozdziale pojęcia i terminy zostały użyte w opisie w dalszej części pracy.

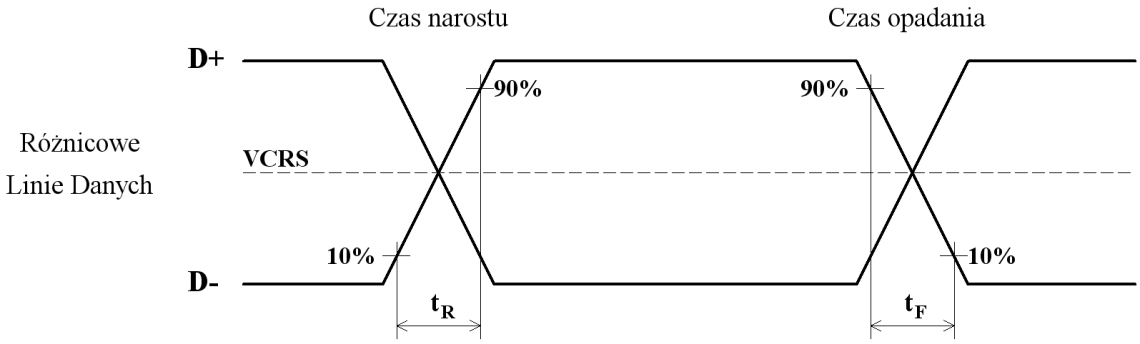
ACK	Odpowiedź na poprawnie przeprowadzoną transmisję.
Bit Stuffing	Wstawianie zera logicznego w ciąg danych, co zapewnia poprawną synchronizację przebiegów zegarowych w odbiorniku.
Buffer	Pamięć używana podczas transmisji między urządzeniami.
Bulk Transfer	Jeden z czterech typów transmisji obsługiwanych przez USB. Jest używany podczas masowej transmisji danych.
Bus Enumeration	Wykrywanie i identyfikacja urządzeń na magistrali USB.
Control Transfer	Jeden z czterech typów transmisji obsługiwanych przez USB. Obsługuje konfigurację, wysyłanie komend oraz pobieranie statusu od urządzenia.
CRC	Cyclic Redundancy Check - cykliczny kod nadmiarowy. Używany jest do kontroli poprawności transmitowanych danych.
Default Address	Domyślny adres (00H) używany przez urządzenie zaraz po podłączeniu do magistrali USB.
Device Address	Siedmiobitowy adres reprezentujący interfejs na magistrali USB, nadawany przez kontroler (Host).

[1] Universal Serial Bus Specification, rev. 2.0

Endpoint	Unikalna adresowalna część urządzenia USB, umożliwiająca komunikację obustronną między kontrolerem a urządzeniem. W dalszej części pracy pod tym terminem rozumiany jest bufor wejścia - wyjścia danych.
EOP	End of Packet - Koniec pakietu danych.
Full-speed	Transmisja danych z prędkością $12Mb/s$.
Host	Kontroler - Komputer zawierający sprzętowo zaimplementowaną magistralę USB i system operacyjny.
LSB	Najmniej znaczący (najmłodszy) bit.
MSB	Najbardziej znaczący (najstarszy) bit.
NAK	Odpowiedź na odrzuconą transmisję danych.
NRZI	Non Return to Zero Invert - Standard kodowania, w którym zero logiczne reprezentowane jest przez zmianę stanu logicznego na przeciwny, a jedynka logiczna przez brak zmiany stanu. Eliminuje konieczność przesyłania przebiegów zegarowych
Packet	Pakiet - Grupa danych zorganizowana w specjalny sposób i przygotowana do wysłania
PID	Packet ID, nagłówek - Pole określające typ przesyłanego pakietu, a co za tym idzie jego format i rodzaj kontroli błędów.
PLL	Phase Locked Loop - Pętla fazowa umożliwiająca synchronizację wewnętrznego układu zegarowego z częstotliwością danych.
Protocol	Protokół - Zbiór zasad określających format i zależności czasowe podczas transmisji między dwoma urządzeniami.
Request	Żądanie będące częścią protokołu konfiguracyjnego interfejsu.
SOP	Start of Packet - Początek pakietu.
Stack	Stos USB. Logiczna część całego interfejsu. Zajmuje się obsługą protokołu komunikacyjnego. Analizuje strukturę otrzymywanych danych oraz przygotowuje pakiety do wysyłki.
Token Packet	Typ pakietu określający jaki rodzaj i kierunek transmisji będzie nawiązywany na magistrali USB.
Transceiver	Sprzętowa część interfejsu USB. Generuje przebiegi zegarowe podczas nadawania sygnału oraz zajmuje się odzyskiwaniem przebiegu zegarowego z przychodzących danych w przypadku odbioru pakietu.

2.3 Parametry sygnału dla standardu Full-Speed (12Mb/s)

Sygnały generowane przez Transceiver'y, zarówno po stronie komputera jak i urządzenia podłączonego do magistrali USB, muszą spełniać określone standardy. Szczególnie ważne są tutaj czasy narastania (t_R) i opadania (t_F) sygnału na liniach danych (D+ i D-). W urządzeniach pracujących w standardzie Full-Speed czasy narastania i opadania sygnału, mierzone między 10% a 90% jego wartości, powinny się zawierać między $4ns$ a $20ns$. Zasadę pomiaru czasów narastania i opadania przedstawiono na rysunku (1).

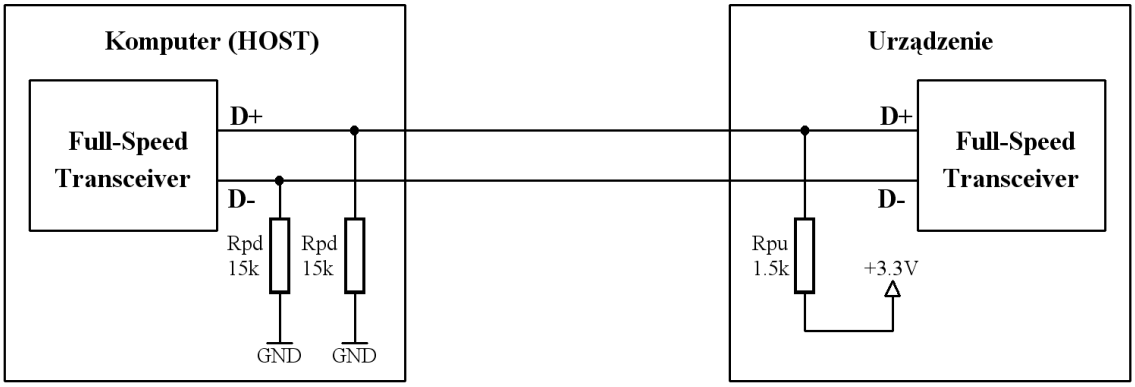


Rysunek 1: Czasy narastania i opadania sygnału na liniach danych

Dla urządzeń pracujących w standardzie Full-Speed średnia wartość napięcia ($VCRS$[2] - Crossover Voltage), wokół którego następuje zmiana poziomów logicznych na liniach danych (D+ i D-) musi się zawierać w przedziale od $1.3V$ do $2.0V$. Specyfikacja uniwersalnej magistrali szeregowej wymaga, aby wartość średnia napięcia $VCRS$ była stała. Pociąga to za sobą warunek jednakowych amplitud sygnałów na obu liniach (D+ i D-) oraz wymusza jednakowe czasy narastania i opadania obu przebiegów.

2.4 Identyfikacja prędkości pracy urządzenia

Jak już wcześniej wspomniano, na magistrali USB mogą pracować urządzenia o różnej prędkości. Rozróżnienie tego, czy urządzenie pracuje w standardzie Low-Speed, czy w standardzie Full-Speed, odbywa się za pomocą rezystora o wartości $1.5k\Omega$ podciągającego linie danych do napięcia $3.0 - 3.6V$. Jeżeli rezystor zostanie dołączony do linii D-, to urządzenie zostanie rozpoznane jako Low-Speed, jeśli natomiast rezystor zostanie dołączony do linii D+, komputer rozpozna urządzenie jako pracujące w standardzie Full-Speed. Na rysunku (2) przedstawiono sposób podłączenia urządzenia do komputera.



Rysunek 2: Identyfikacja urządzenia jako pracujące w standardzie Full-Speed

Od strony komputera (Hosta) zwykle stosuje się rezystory o wartości $15k\Omega$, "ściągające" linie danych do masy(GND). Wartości wszystkich rezystorów muszą mieć podane wartości z dokładnością do 5%. Powodują one ustalenie się na liniach danych potencjałów zgodnych ze

[2]Universal Serial Bus Specification, rev. 2.0, s. 130

standardami zawartymi w specyfikacji USB.

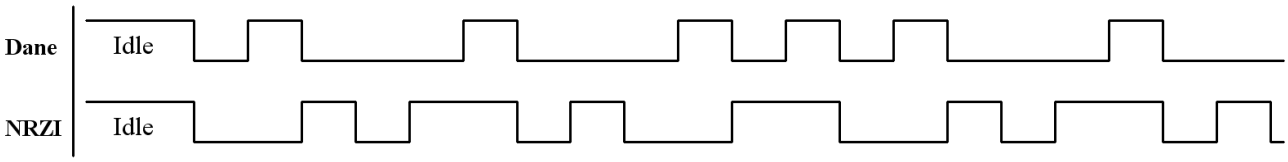
Dzięki rezystorom występującym w torze sygnałowym stany logiczne na liniach D+ i D- przyjmują wartości napięć z przedziału $0.8 - 2.5V$. W specyfikacji USB zostały zdefiniowane specjalne różnicowe[3] stany logiczne, które w standardzie Full-Speed przedstawiają się następująco:

$$\text{Stan K: (różncowo: "0")} \quad \text{gdy} \quad (D-) - (D+) > 200mV$$

$$\text{Stan J: (różncowo: "1")} \quad \text{gdy} \quad (D+) - (D-) > 200mV$$

2.5 Kodowanie NRZI

Kodowanie NRZI (Non Return to Zero Invert) jest wykorzystywane przez USB do transmisji danych. W kodowaniu NRZI wysoki stan logiczny (jedynka) jest reprezentowany przez brak zmiany stanu logicznego, natomiast zero reprezentowane jest zmianą stanu logicznego na przeciwny. Ciąg zakodowanych zer będzie zatem ciągłą zmianą stanu logicznego, natomiast ciąg jedynek logicznych wygeneruje długie okresy w których stan się nie zmienia. Zasadę kodowania NRZI przedstawiono na rysunku (3)



Rysunek 3: Kodowanie danych do standardu NRZI

Kodowanie NRZI pozwala uniknąć przesyłania przebiegu zegarowego równolegle do linii danych. Sygnał zegarowy zostaje zakodowany razem z danymi w jednym sygnale. Takie działanie pozwala uniknąć przesunięć fazowych powstałych między danymi a sygnałem zegarowym, ale wymaga w odbiorniku układu odzyskiwania przebiegu zegarowego, tworzącego przebieg o odpowiedniej częstotliwości i fazie na podstawie zmian sygnału danych (NRZI).

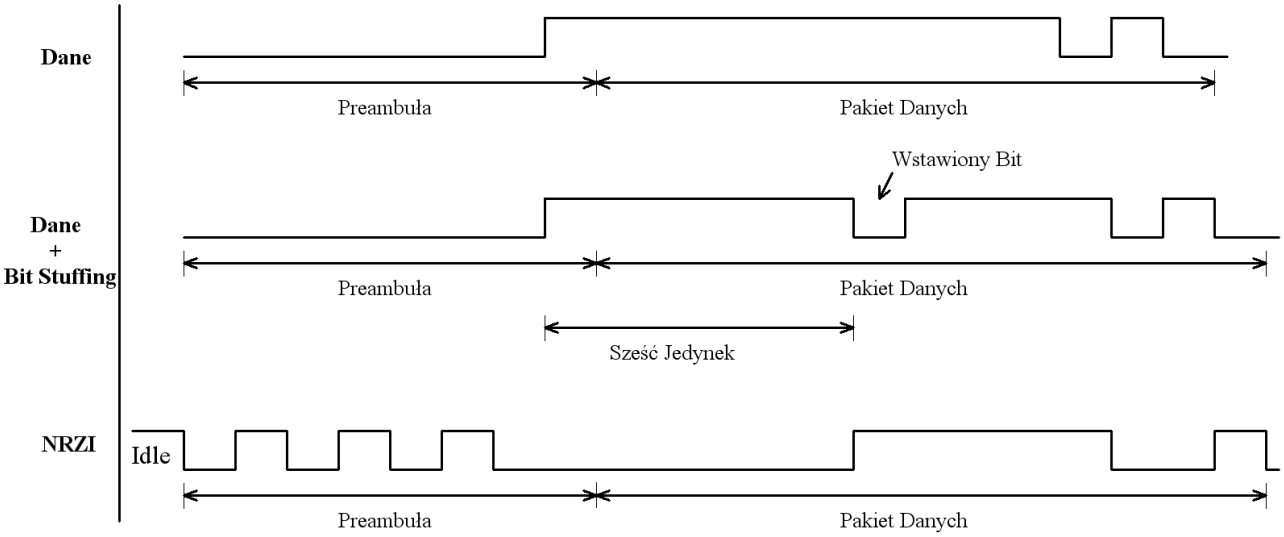
2.6 Bity synchronizacyjne - Bit Stuffing

Ponieważ transmisja danych po uniwersalnej magistrali szeregowej odbywa się bez przesyłania przebiegu zegarowego, zostaje on wygenerowany w odbiorniku na podstawie zmian w sygnale danych (jest synchronizowany każdą zmianą sygnału danych, aby był zgodny z nimi w fazie).

Gdyby zdarzyło się, że dane zawierają same jedynki logiczne (brak zmiany stanu logicznego), mogłoby dojść do przesunięć fazowych między tworzonym w odbiorniku przebiegiem zegarowym, a przychodzącymi danymi. Aby uniknąć tego problemu wprowadzono dodatkowy mechanizm wstawiania bitów synchronizacyjnych(Bit Stuffing[4]), polegający na wstawianiu zera logicznego po każdych sześciu, bezpośrednio po sobie następujących, jedynkach logicznych.

[3]Universal Serial Bus Specification, rev. 2.0, Tabela 7-2, s. 145

[4]Universal Serial Bus Specification, rev. 2.0, s. 157



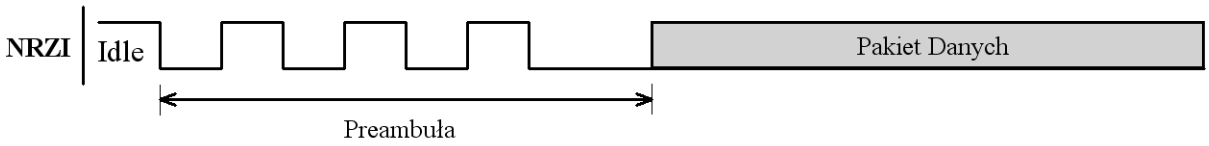
Rysunek 4: Zasada wstawiania bitów synchronizacyjnych

Gwarantuje to poprawne odzyskiwanie i synchronizację przebiegu zegarowego. Zasadę wstawiania bitów synchronizacyjnych przedstawiono na rysunku (4)

Wstawianie bitów synchronizacyjnych jest aktywne od początku nadawania preambuły (opis w podrozdziale 2.7). Jedynka logiczna kończąca preambułę także jest zliczana jako pierwsza w sekwencji sześciu kolejnych. Bity synchronizacyjne wstawiane są zawsze. Nawet gdy pakiet danych kończy się sześcioma jedynkami i tak zostanie wstawione zero, bezpośrednio przed sygnałem końca transmisji. Odbiornik, po rozkodowaniu transmisji w standardzie NRZI do postaci zwykłego sygnału logicznego, musi rozpoznać wstawione bity i usunąć je z ciągu danych. Jeżeli w sygnale zostanie wykrytych siedem jedynek logicznych, cały pakiet musi zostać zignorowany.

2.7 Preambuła - Bajt synchronizacyjny

Preambuła (Sync Pattern[5]) jest bajtem synchronizacyjnym złożonym z siedmiu zer zakończonych jedynką logiczną. Widok sygnału preambuły w kodowaniu NRZI przedstawiono na rysunku (5).



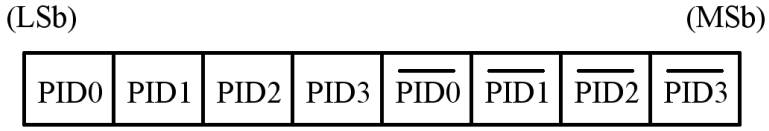
Rysunek 5: Wygląd sygnału preambuły w kodowaniu NRZI

Sygnał ten nadawany jest na początku każdej transmisji. Na jego podstawie (ciągłe zmiany stanów logicznych) odbiornik dostraja swój sygnał zegarowy, tak aby zgadzał się w fazie z przychodzącymi danymi.

[5]Universal Serial Bus Specification, rev. 2.0, s. 159

2.8 Nagłówek pakietów (PID)

Nagłówek pakietów (Packet ID) pojawiający się zaraz po polu preambuły, jest częścią każdego pakietu transmitowanego po magistrali USB. PID zawiera czterobitowe pole typu pakietu, po którym występuje pole testowe złożone z zanegowanych wartości tych bitów. Strukturę pola nagłówka pakietów przedstawiono na rysunku (6).

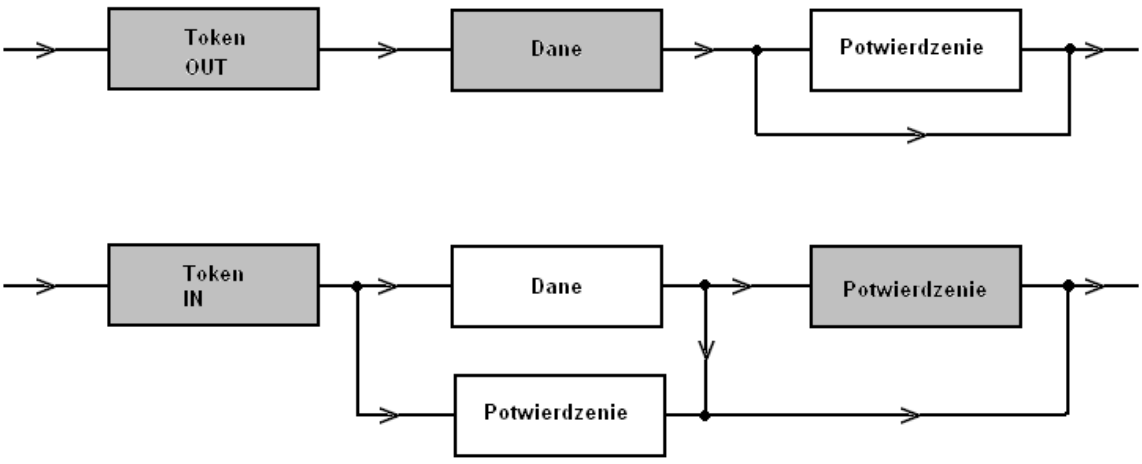


Rysunek 6: Struktura nagłówka pakietów

PID zawiera informację o typie pakietu, jego formacie i rodzaju kontroli błędów, jaka musi zostać przeprowadzona. Zanegowana część nagłówka pakietów umożliwia kontrolę jego poprawności i gwarantuje, że odbierany pakiet zostanie poprawnie zinterpretowany. Jeśli pierwsze cztery bity nie zgadzają się z negacją czterech dalszych, cały pakiet jest ignorowany. Tak samo się dzieje jeżeli urządzenie odbierze poprawny PID, lecz jego typ nie jest obsługiwany. Dokładny opis wszystkich nagłówków[6] dostępny jest w specyfikacji USB.

2.9 Struktura protokołów transmisji USB

Grupy pakietów połączonych w większe bloki logiczne nazwano protokołami transmisji (w miejsce oryginalnego angielskiego terminu 'Transmission Transacion'). Interfejs obsługuje protokół wymiany rozkazów i danych konfiguracyjnych ('Control Transfers')[7] oraz protokół nadawania ('In') i odbioru ('Out') danych w trybie transmisji masowej ('Bulk Transactions')[8]. Tryb ten wybrano ze względu na najbardziej rozbudowany mechanizm kontroli poprawności transmisji.

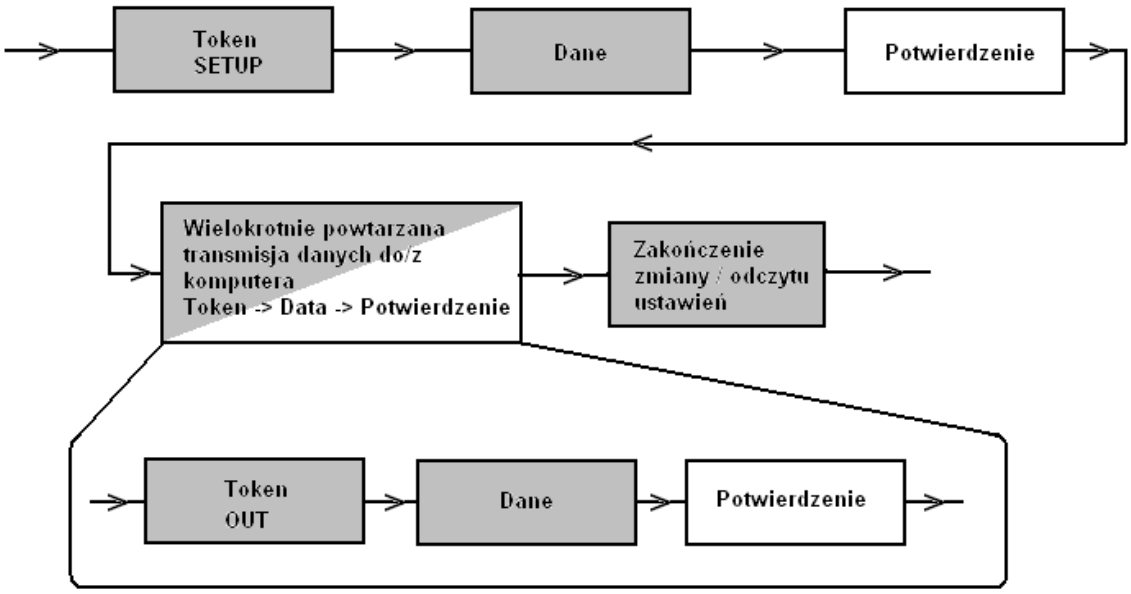


Rysunek 7: Protokoły nadawania i odbierania danych

[6]Universal Serial Bus Specification, rev. 2.0, s. 196
[7]Universal Serial Bus Specification, rev. 2.0, s. 225
[8]Universal Serial Bus Specification, rev. 2.0, s. 221

Na rysunku (7) przedstawiono schematycznie protokoły nadawania i odbierania danych. Wymianę danych rozpoczyna generowany przez komputer pakiet typu 'Token', o podtypie determinującym kierunek transmisji ('IN' dla danych nadawanych przez interfejs i 'OUT' dla odbieranych). Oprócz tej informacji, pakiet zawiera również siedmiobitowy adres urządzenia, z którym nawiązywana jest transmisja, oraz czterobitowy numer bufora z którym komputer chce się skomunikować. Przy transmisji skierowanej do interfejsu, komputer bezpośrednio po pakiecie 'Token' generuje pakiet typu 'Data' zawierający maksymalnie 64 bajty danych. W odpowiedzi interfejs wysyła pakiet potwierdzenia typu 'Handshake', albo informujący komputer o poprawnie (podtyp 'ACK') przeprowadzonym transferze danych, albo, w przypadku niemożności odebrania danych (bufor wyjściowy zapełniony), zawiadamiający o błędzie odbioru pakietu (podtyp 'NAK'). Wystąpienie przekłamania transmisji (niezgodność cyklicznego kodu nadmiarowego CRC) jest komunikowane poprzez brak jakiejkolwiek odpowiedzi na pakiet 'Data'.

Jeżeli komputer wywoła protokół nadawania danych poprzez pakiet typu 'Token IN', interfejs ma możliwość wysłania danych zgromadzonych w buforze wejściowym. W tym celu interfejs nadaje pakiet typu 'Data' bezpośrednio po odebranym wezwaniu 'Token IN'. Komputer, po odebraniu danych, informuje interfejs o powodzeniu przebiegu transmisji za pomocą pakietu 'Handshake ACK'. Błąd, podobnie jak przy odbieraniu danych, sygnalizowane jest brakiem pakietu potwierdzenia 'Handshake'. W przypadku, gdy interfejs nie ma żadnych danych do wysłania, sygnalizuje to generując pakiet 'Handshake NAK' w miejsce pakietu 'Data'.



Rysunek 8: Protokół konfiguracyjny

Rysunek (8) przedstawia protokół konfiguracyjny. Jest on inicjowany poprzez wysyłany z komputera pakiet 'Token SETUP' za którym podąża pakiet 'Data' zawierający rozkazy konfiguracyjne (m.in. kierunek dalszej fazy transmisji - rozkazów bądź ustawień ze strony komputera lub opisów interfejsu i jego konfiguracji do komputera). Analogicznie do transmisji danych, faza ta wymaga potwierdzenia odpowiednim pakietem 'Handshake'. W kolejnej fazie następuje wymiana danych konfiguracyjnych podlegająca strukturze transmisji danych. Protokół konfiguracyjny kończy pakiet 'Token' o kierunku transmisji (podtyp 'OUT' lub 'IN') przeciwnym do zadeklarowanego w rozkazie konfiguracyjnym na początku protokołu. Następnie komputer (przy fazie kończącej rozpoczętej przez 'Token OUT') lub interfejs (dla 'Token IN') generuje

tzw. 'Data Zero Lenght', czyli pakiet typu 'Data' zawierający bezpośrednio po nagłówku szesnastobitowy cykliczny kod nadmiarowy CRC16 (innymi słowy o zerowej długości bloku danych). Otrzymanie potwierdzenia 'Handshake ACK' oznacza zakończenie protokołu konfiguracyjnego.

Pakiety typu 'Data' są wyposażone w dodatkowy mechanizm zabezpieczający przed błędną retransmisją poprawnie odebranych danych. Sytuacja taka może nastąpić, gdy nadawca pakietu 'Data' nie odbierze wygenerowanego przez odbiorcę protokołu potwierdzenia 'Handshake ACK'. Wówczas odbiorca uważa pakiet danych za poprawny i przekazuje go do dalszych operacji, podczas gdy nadawca, nie otrzymując potwierdzenia, uzna transmisję za przerwaną i wyśle ponownie pakiet 'Data', zawierający znów te same dane. Aby ustrzec się przed takim błędem, dane wysyłane są w pakietach o dwóch naprzemiennych podtypach - 'DATA0' i 'DATA1'. Odbiorca otrzymując dwa razy z rzędu ten sam podtyp pakietu 'Data' powinien potwierdzić jego otrzymanie ('Handshake ACK'), lecz dane w nim zawarte odrzucić. Wyjątkowymi sytuacjami są faza inicjacji protokołu konfiguracyjnego, gdzie rozkazy konfiguracyjne przesyłane są zawsze w pakiecie podtypu 'DATA0' oraz faza końcowa tego protokołu, gdzie 'Data Zero Lenght' jest zawsze podtypu 'DATA1'.

2.10 Enumeracja interfejsu USB

Enumeracją interfejsu nazywane są kolejne etapy konfiguracji wstępnej interfejsu następujące po jego podpięciu do magistrali USB[9]. Proces enumeracji można podzielić na cztery główne etapy opisane poniżej.

1. Interfejs wykrywa dołączenie do magistrali dzięki detekcji napięcia +5V, przechodząc w pierwszy stan enumeracji - zasilania. Za wykrywanie podłączenia urządzenia do magistrali USB odpowiedzialny jest zewnętrzny układ wykonawczy opisany w rozdziale 8;

2. Sygnał resetu magistrali, generowany przez komputer, powoduje przejście do ustawień domyślnych interfejsu (domyślny adres, opróżnienie buforów danych, itp.). Do czasu zakończenia czwartego etapu enumeracji, każdy sygnał resetu magistrali powoduje skasowanie ustawień i powrót interfejsu do stanu standardowego;

3. Interfejs poprzez protokół konfiguracyjny otrzymuje od komputera unikalny adres. Od tego momentu (stan zaadresowania) odpowiada tylko na pakiety kierowane pod otrzymany adres;

4. Po otrzymaniu opisu możliwych konfiguracji interfejsu komputer wybiera jedną z nich. Odebranie przez interfejs protokołu ustawiającego wybraną konfigurację oznacza zakończenie procesu enumeracji i przejście do stanu skonfigurowania.

[9]Universal Serial Bus Specification, rev. 2.0, s. 241

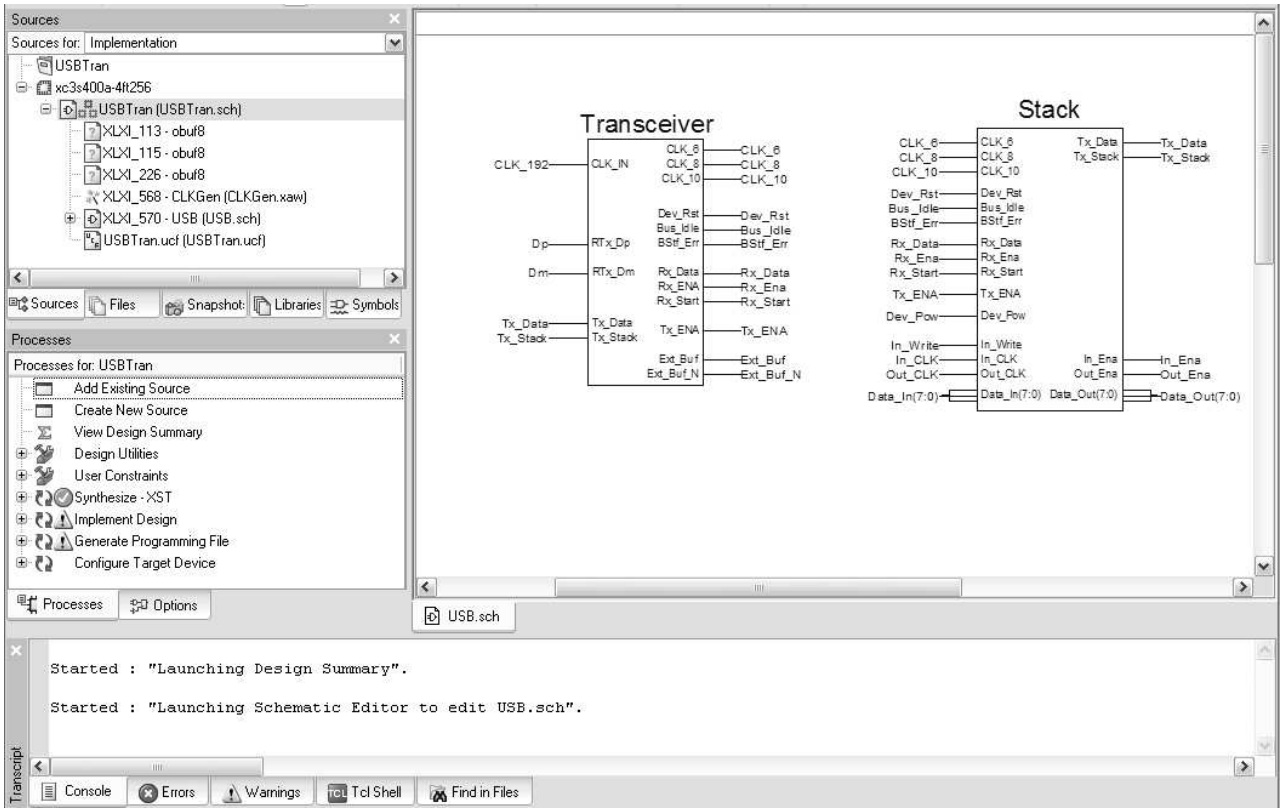
3 Projektowanie i Implementacja

Projekt układu został w całości opracowany na podstawie specyfikacji USB i zaimplementowany w układzie programowalnym (FPGA - Field Programmable Gate Array). Logikę działania projektowanego w ramach pracy interfejsu USB wykonano w programie ISE 10.1, udostępnionym przez firmę Xilinx. Schematy tworzone w tym programie zostały poddane syntezie logicznej, po czym załadowane fizycznie do układu FPGA. Symulacje działania tworzonego układu okazały się zbędne, gdyż wynik jego działania był bezpośrednio obserwowany w fizycznym układzie.

Układem FPGA, na którym zaimplementowano interfejs USB, jest XC3S400A (Spartan-3A) firmy Xilinx zamontowany na płytce testowej AES-SP3A-EVAL400-G (Xilinx Spartan-3A Evaluation Kit) firmy Avnet.

3.1 Środowisko Programistyczne - Xilinx ISE 10.1

Xilinx ISE[10] jest zintegrowanym środowiskiem programistycznym służącym do przeprowadzania wszystkich etapów tworzenia projektu układu cyfrowego, aż do jego implementacji wewnątrz układu FPGA. Okno główne programu przedstawiono na rysunku (9).



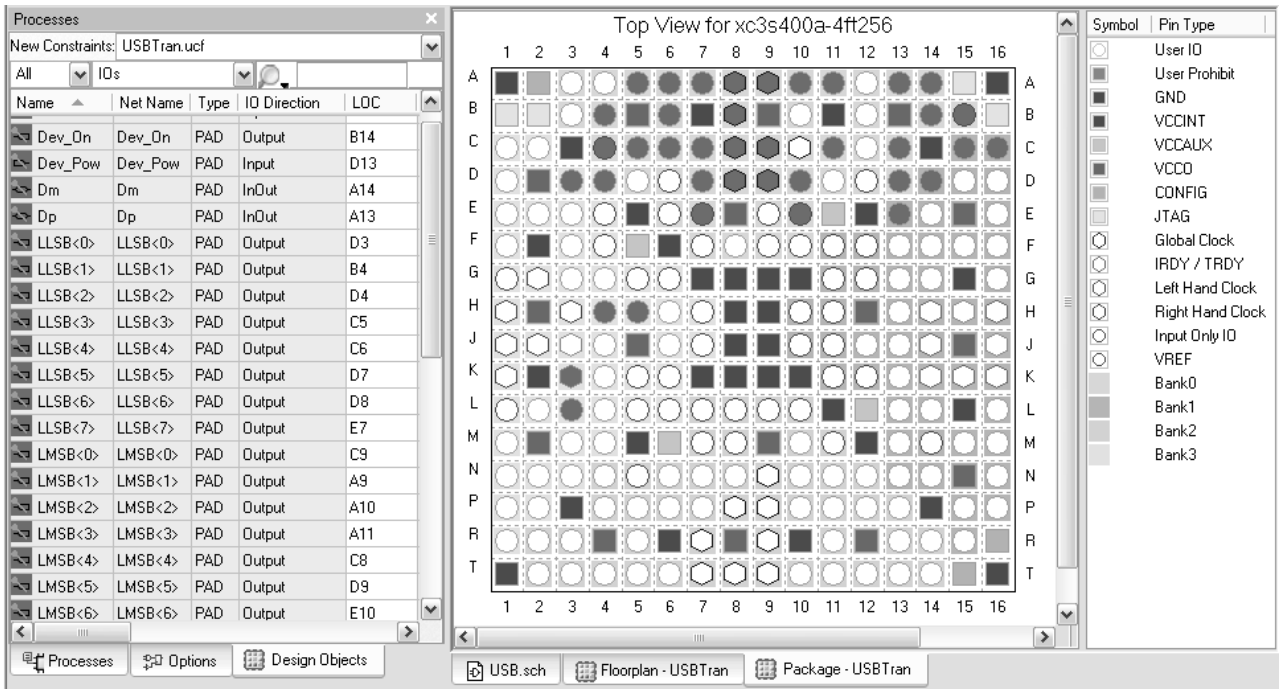
Rysunek 9: Wygląd okna głównego programu ISE 10.1

Program umożliwia edycję plików projektu, sterowanie procesem syntezy logicznej, ustawianie parametrów użytkownika (takich jak wyprowadzenia fizyczne i stałe czasowe), sterowanie procesem implementacji i bezpośrednie zaprogramowanie układu FPGA, lub wygenerowanie

[10] http://direct.xilinx.com/direct/ise10_tutorials/ise10tut.pdf, dokładny opis programu ISE 10.1

pliku konfiguracyjnego dla układu programowalnego.

W pracy, podczas tworzenia interfejsu USB, korzystano z opcji 'Generate Programming File' która powoduje utworzenie pliku konfiguracyjnego dla układu FPGA. Gotowy plik ładowano do pamięci układu programowalnego za pomocą dodatkowego programu (AvProg) dostarczonego przez producenta płytki testowej. Użycie dodatkowego programu umotywowane jest konstrukcją samej płytki, która posiada swój specjalny interfejs programujący.



Rysunek 10: Ustawienia wyprowadzeń układu FPGA

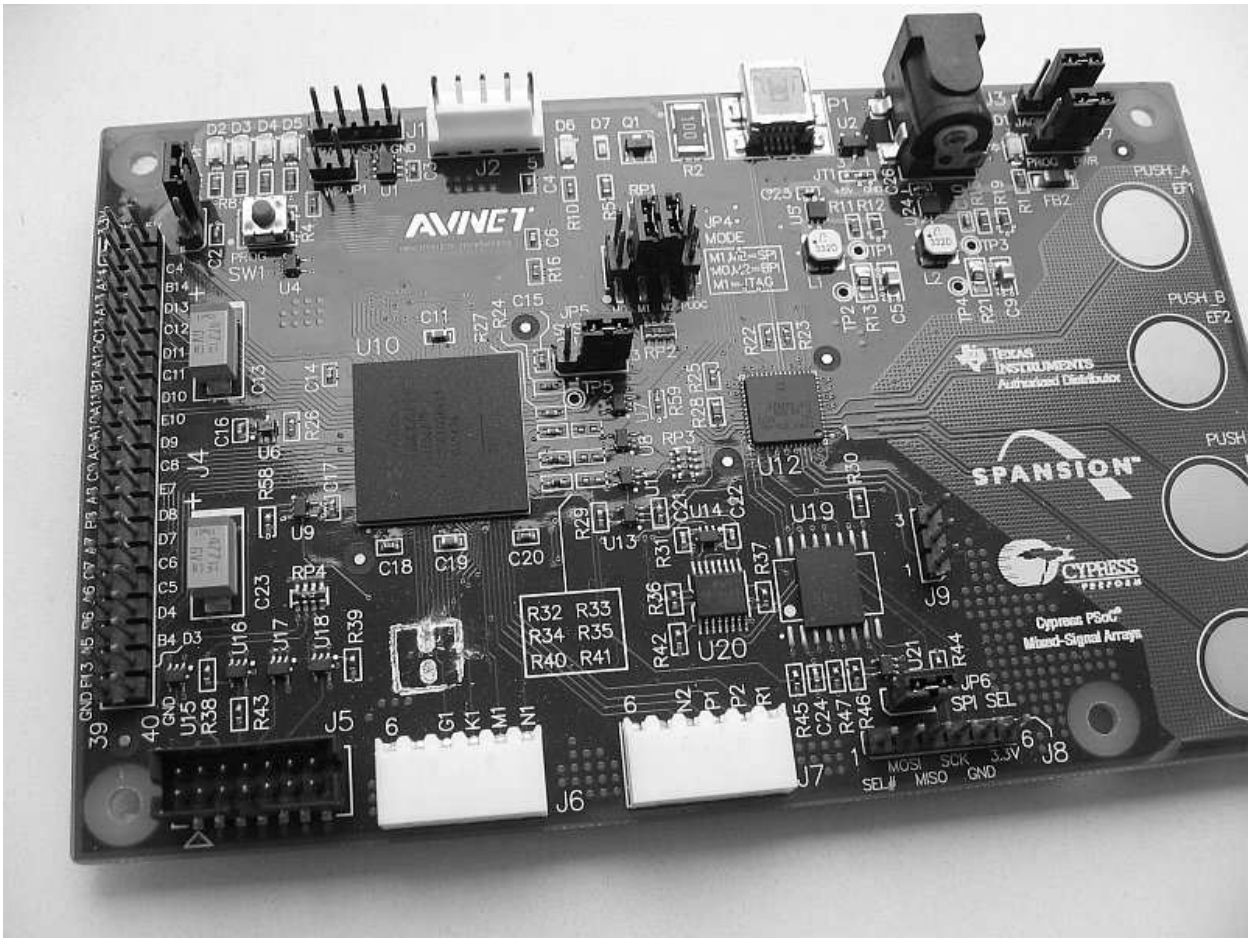
Bardzo ważnym etapem projektowym jest przypisanie sygnałów wejściowych i wyjściowych, występujących w projekcie, do konkretnych wyprowadzeń układu scalonego. Jest to fragment szerszego zagadnienia jakim jest ustawianie ograniczeń projektowych 'User Constraints'. Ustawienia te przechowywane są w pliku $*.ucf$. Na rysunku (10) przedstawiono wygląd edytora do ustawiania wyprowadzeń układu FPGA. Możliwa jest także bezpośrednia edycja ustawień w pliku $*.ucf$

Środowisko programistyczne Xilinx ISE 10.1, oprócz projektowania i edycji schematów ideowych, udostępnia także wiele narzędzi programistycznych, takich jak kompilatory dla języków Verilog i VHDL czy edytor diagramów stanów. W pracy przyjęto metodę projektową w formie edycji schematów ideowych, gdyż taka okazała się najwygodniejsza. Dzięki schematom ideowym wyjaśnienie działania projektowanego w ramach pracy interfejsu USB, jest najprostsze.

3.2 Płytka Testowa dla Xilinx Spartan-3A

Sam układ programowalny nie może działać bez współpracujących elementów zewnętrznych. Zostały one zamontowane razem z nim na płytce drukowanej, którą ze względu na przeznaczenie nazwano płytką testową. Układem FPGA pracującym w tym środowisku testowym jest XC3S400A (Spartan-3A) firmy Xilinx. Zdjęcie płytki testowej używanej podczas projektowania

interfejsu USB przedstawiono na rysunku (11).



Rysunek 11: Wygląd płytki testowej AES-SP3A-EVAL400-G (Xilinx Spartan-3A Evaluation Kit) używanej podczas projektowania

Na płytce testowej znajdują się podstawowe elementy umożliwiające utrzymanie napięcia zasilania na odpowiednim poziomie i zapewniające jego stabilność. Dzięki elementom dodatkowym, takim jak diody LED czy przyciski, możliwa jest prosta interakcja budowanego układu logicznego z użytkownikiem.

Ważnym elementem znajdującym się na płytce testowej jest generator zegarowy o częstotliwości $16MHz$. Niestety na potrzeby projektowanego interfejsu USB zastosowano zewnętrzny kwarcowy generator zegarowy, ponieważ ten dostępny na płytce nie posiada wystarczającej stabilności częstotliwości. Do komunikacji z układami zewnętrznymi przeznaczone jest specjalne złącze, do którego doprowadzone są bezpośrednio wyprowadzenia zewnętrzne układu FPGA.

Do programowania znajdującego się na płytce układu Spartan-3A przeznaczony jest specjalny interfejs programujący. Główną jego częścią jest konwerter (mostek) USB → UART(RS-232), dzięki czemu programowanie jest szybkie i wygodne za pomocą programu dołączonego przez producenta płytki, firmę Avnet[11]. Płytka testowa jest bardzo wygodna w użyciu, gdyż umożliwia przeprowadzenie testów na fizycznym układzie i zwalnia z konieczności symulacji.

[11] http://em.avnet.com/spartan3a-evl, strona firmy Avnet z informacjami na temat płytki testowej

4 Transceiver USB, Opis ogólny

4.1 Wstęp

Transceiver jest sprzętową częścią interfejsu uniwersalnej magistrali szeregowej (USB). Jego zadaniem jest generowanie przebiegów zegarowych podczas nadawania sygnału do komputera oraz odzyskiwanie przebiegu zegarowego z przychodzących danych w przypadku odbioru transmisji z komputera. Bardzo ważnym zadaniem Transceiver'a jest wstępna obróbka przychodzących danych polegająca na odcinaniu sygnału synchronizacji (tak zwanej preambuły) w przypadku odbioru pakietu, oraz generacja tego sygnału podczas jego wysyłania. Kolejnym zadaniem części sprzętowej jest wykrywanie sygnałów końca transmisji gdy komputer nadaje, oraz generacja takiego sygnału gdy urządzenie kończy nadawanie. Dzięki wycinaniu i wstawianiu bitów synchronizacyjnych, tak zwanego Bit-Stuffing'u (dodatkowe zero w pakiecie danych nie będące jego częścią a konieczne do poprawnej synchronizacji przebiegu zegarowego) oraz wszystkich pozostałych wstępnych operacji, część logiczna całego interfejsu (tak zwany stos) dostaje "czysty" pakiet danych i zajmuje się jedynie analizą protokołu komunikacyjnego uniwersalnej magistrali szeregowej.

4.2 Działanie układu

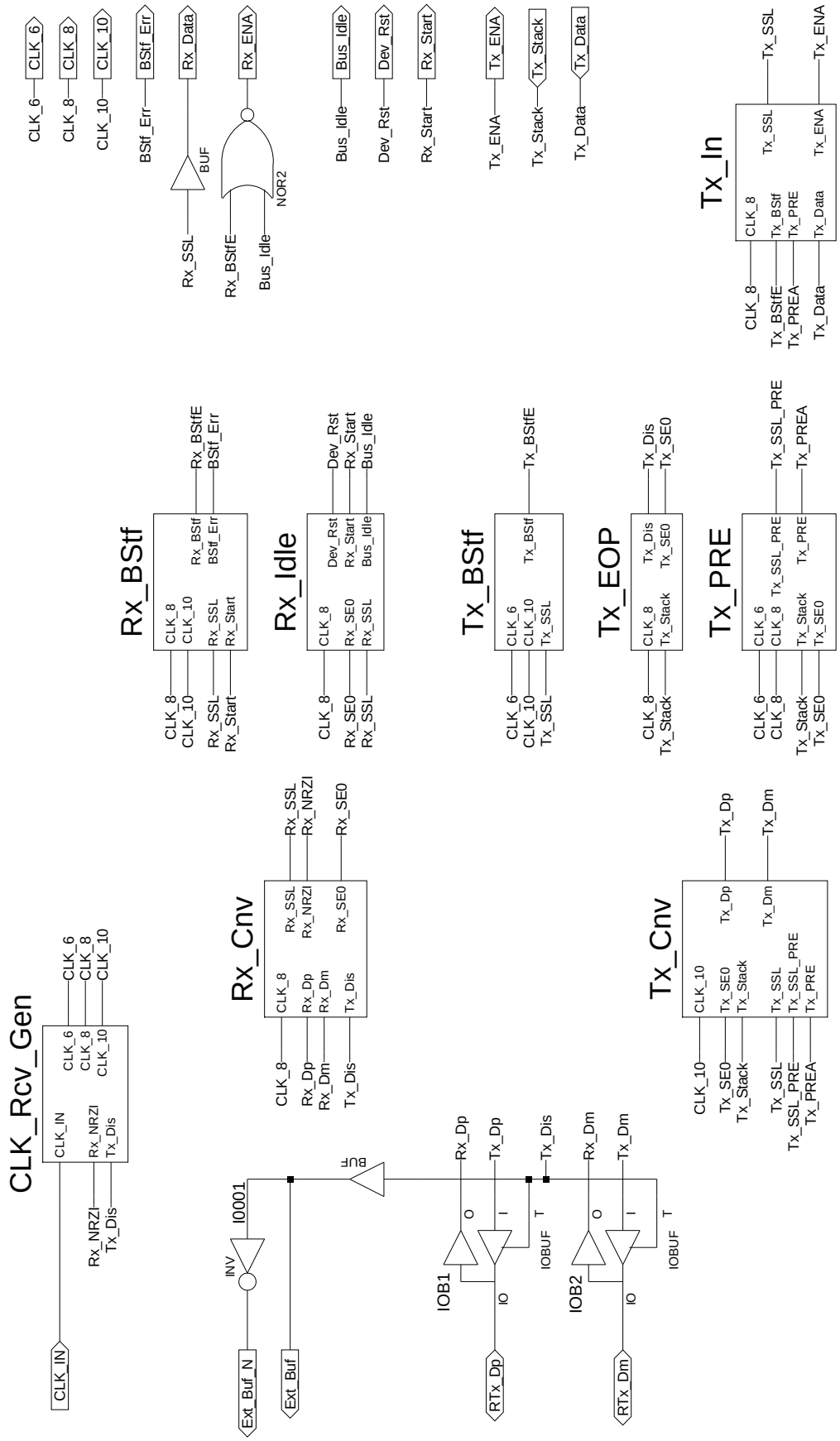
Schemat blokowy Transceiver'a znajduje się na rysunku (12). W celu zwiększenia czytelności schematu zrezygnowano z połączeń przewodowych między blokami, w miejsce których wprowadzono specjalne nazewnictwo połączeń (linie o tych samych nazwach są ze sobą połączone).

Dwukierunkowy sygnał danych pochodzący z komputera (linie *RTx_Dp* i *RTx_Dm*) trafia na bufory IOB1 i IOB2, będące elementem pośredniczącym między logiką wewnętrzną a fizycznymi wyprowadzeniami układu scalonego. Bardzo ważnym zadaniem tych buforów jest połączenie ze sobą obwodów wejściowych odbiornika i obwodów wyjściowych nadajnika. Umożliwia to odczyt stanów logicznych z linii *RTx_Dp* i *RTx_Dm* gdy nadajnik jest wyłączony i ustawianie stanów logicznych na tych liniach podczas nadawania. Włączenie nadajnika następuje poprzez wystawienie stanu niskiego na wejściu sterującym buforów. Ponieważ część buforów odpowiedzialna za odczyt danych jest cały czas aktywna i podczas nadawania urządzenie odbierałoby swoją transmisję, wykorzystano sygnał sterujący buforów do dezaktywacji odbiornika (linia *Tx_Dis*). Dodatkowo *Tx_Dis* wyprowadzony jest na zewnątrz układu Transceiver'a w postaci sygnału *Ext_Buf*, gdzie został użyty do sterowania zewnętrznego bufora, układu scalonego HCT244. Ponieważ zewnętrzny układ sprzętowy wymaga także zanegowanego sygnału sterującego, za pomocą inwertera *I0001* został on wyprowadzony jako *Ext_Buf_N*

Ważnym blokiem funkcjonalnym całego interfejsu USB jest układ odzyskiwania i generowania przebiegów zegarowych (**CLK_Rcv_Gen**). Zajmuje się on generacją zespołu przesuniętych w fazie sygnałów zegarowych wykorzystywanych w całym układzie Transceiver'a oraz stosie. Podczas transmisji w kierunku do urządzenia układ ten synchronizuje wewnętrzny zegar na każdej zmianie sygnału danych, zapewniając prawidłowy jego odczyt, bez przesunięć fazowych.

4.2.1 Odbiornik

Odbiornik jest włączony gdy linia *Tx_Dis* przyjmuje stan wysoki. Dane za pośrednictwem buforów IOB1 i IOB2 trafiają do bloku konwertera **Rx_Cnv**, gdzie sygnał NRZI pochodzący z komputera zamieniany jest na zwykły sygnał logiczny (wyjście *Rx_SSL*) oraz wykrywany



Rysunek 12: Schemat blokowy Transceivera USB

jest sygnał końca transmisji czego objawem jest stan wysoki na *Rx_SE0*. Na podstawie tego sygnału, generowanego przez konwerter, kolejny blok funkcjonalny (**Rx_Idle**) wystawia zbiór sygnałów sterujących interpretowanych dalej w logicznej części interfejsu (stosie). Do sygnałów tych należą między innymi: *BUS_Idle* - informujący stos o braku jakiejkolwiek transmisji na liniach danych portu USB, *Dev_Rst* - ustawiany kiedy zostanie wykryta konieczność zresetowania urządzenia oraz *Rx_Start* - będący w stanie wysokim przez jeden cykl zegarowy, potrzebny do ustawienia warunków początkowych w logice stosu na początku każdej transmisji do urządzenia. Układ **Rx_Idle** pełni jeszcze jedną ważną funkcję. Po przełączeniu odbiornika w stan wstrzymania (ustawieniu stanu wysokiego na *BUS_Idle*), co dzieje się na końcu każdej transmisji, odbiornik ponownie włączany jest dopiero po wykryciu końca preambuły. Pozwala to bez trudu uchronić dalszą część układu przed niepotrzebnym już sygnałem synchronizacji.

Ostatnim blokiem odbiornika jest układ wycinania bitów synchronizacyjnych, tak zwanego Bit-Stuffing'u (**Rx_BStf**). Monitoruje on cały czas dane na wyjściu *Rx_SSL* konwertera i w przypadku wystąpienia pod rząd sześciu jedynek wystawia wysoki stan logiczny na wyjściu *Rx_BStf* na jeden cykl zegara (sygnał ten wykorzystywany jest później do usuwania z danych dodatkowego zera, wysyłanego przez nadajnik po każdych sześciu jedynkach w celach synchronizacji nadajnika z odbiornikiem). W efekcie za pośrednictwem bramki logicznej G0001 tworzony jest sygnał zezwolenia na przepisywanie danych (*Rx_ENA*), którego stan wysoki świadczy o właściwych danych na wyjściu *Rx_Data* (*Rx_SSL*). Inaczej mówiąc informacje z linii *Rx_Data* są przepisywane do dalszej obróbki tylko wtedy gdy na *Rx_ENA* jest stan wysoki. Taki mechanizm pozwala na wycinanie bitów synchronizacyjnych podczas właściwej transmisji i chroni przed odczytem samych "jedynek" podczas gdy nic nie jest wysyłane po magistrali i linie danych znajdują się w stanie stabilnym.

4.2.2 Nadajnik

Nadawanie rozpoczyna się w momencie wystawienia przez logikę sterującą (stos) sygnału wysokiego na linii *Tx_Stack*. Za pośrednictwem układu generacji końca transmisji (**Tx_EOP**) ustawiany jest stan niski na linii *Tx_Dis*, co dezaktywuje odbiornik i umożliwia nadawanie przez włączenie wejść buforów IOB1 i IOB2 (Układ generacji końca transmisji mimo swojej nazwy zajmuje się dodatkowo wykrywaniem początku transmisji, chęci nadawania danych przez logikę stosu). W tym samym czasie blok **Tx_PRE** rozpoczyna generację sygnału preambuły, która pojawia się na wyjściu *Tx_SSL_PRE*. Dodatkowo sygnał wysoki na linii *Tx_PRE* informuje pozostałe układy nadajnika o tym, że preambuła jest właśnie generowana.

Drugim ważnym blokiem funkcjonalnym nadajnika jest układ wejściowy (**Tx_In**). Jego zadaniem jest pobieranie kolejnych bitów danych z układu logiki sterującej (stosu). Dane do wysłania wystawiane są przez stos na linii *Tx_Data* (pierwsza wartość wystawiana jest równocześnie z sygnałem *Tx_Stack*) a następnie przepisywane na wyjście *Tx_SSL* układu **Tx_In** przy każdym zboczu narastającym przebiegu zegarowego. Wysoki stan logiczny na wyjściu *Tx_ENA* informuje stos o konieczności wystawienia na linię *Tx_Data* kolejnego bitu danych, na najbliższym zboczu narastającym sygnału zegarowego. Sygnał *Tx_ENA* nie jest ustawiany w stan wysoki dopóki linia *Tx_PRE* nie będzie w stanie niskim co mówi o końcu generowania preambuły.

Kolejnym blokiem nadajnika jest układ wstawiania bitów synchronizacyjnych, tak zwanego Bit-Stuffing'u (**Tx_BStf**). Po odliczeniu sześciu kolejnych "jedynek" znajdujących się w sygnale *Tx_SSL* układ ten wystawia stan wysoki na wyjściu *Tx_BStf*. Zapewnia to ustawienie w układzie **Tx_In** sygnału *Tx_ENA* na poziom niski, a tym samym wstrzymuje na jeden cykl

zegarowy pobieranie kolejnych bitów danych ze stosu. W tym samym czasie linia *Tx_SSL* jest ustawiana w stan niski, co generuje dodatkowe zero logiczne nie należące do danych ale wymagane do poprawnej synchronizacji nadajnika z odbiornikiem.

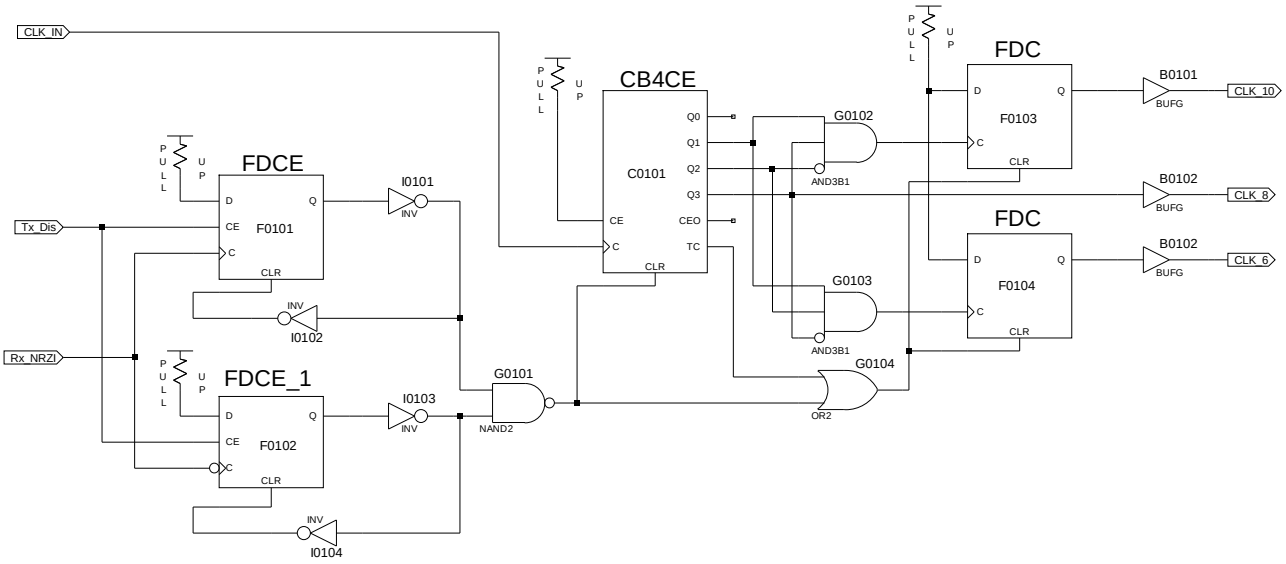
Ważnym blokiem w nadajniku jest układ konwertera (**Tx_Cnv**). Zapewnia on zamianę zwykłego sygnału logicznego pochodzącego z *Tx_In* lub *Tx_PRE* na sygnał w standardzie NRZI używany przy transmisji na uniwersalnej magistrali szeregowej.

Ostatnim blokiem funkcjonalnym nadajnika jest układ generacji końca transmisji (**Tx_EOP**). Po wystawieniu przez stos stanu wysokiego na linii *Tx_Stack* (chęć nadawania), zostaje ustawiony stan niski na linii *Tx_Dis* (sterowanie wewnętrznymi buforami dwukierunkowymi). Gdy sygnał na linii *Tx_Stack* opada (koniec danych) układ **Tx_EOP** wystawi stan wysoki na wyjściu *Tx_SE0* trwający dwa cykle zegarowe. Na podstawie tego konwerter **Tx_Cnv** ustawi dwie linie danych USB w stan niski co poinformuje komputer że urządzenie skończyło nadawać. Po powrocie sygnału *Tx_SE0* do stanu niskiego zostanie jednocześnie odłączony układ nadajnika a włączony odbiornik

5 Opis działania bloków funkcjonalnych Transceiver'a

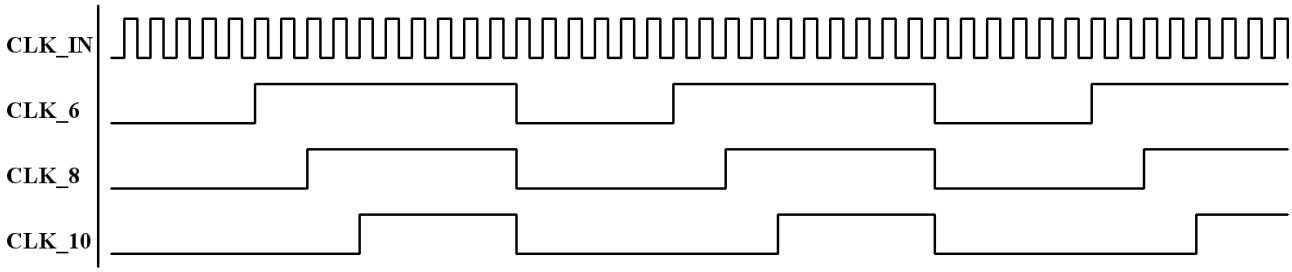
5.1 Blok odzyskiwania przebiegu zegarowego (CLK_Rcv_Gen)

Schemat ideowy **CLK_Rcv_Gen** przedstawiony jest na rysunku (13). Przebieg zegarowy o częstotliwości $192MHz$ podawany jest na wejście *CLK_IN*. Taką częstotliwość uzyskujemy dzięki pomnożeniu przebiegu wyjściowego z generatora kwarcowego ($16MHz$) przez 12. Użycie specjalnego generatora kwarcowego okazało się konieczne ze względu na wymaganą dużą stabilność częstotliwości. Dalej sygnał o częstotliwości $192MHz$ trafia na wejście zegarowe czterobitowego licznika binarnego C0101, dzielącego tą częstotliwość przez 16. Takie rozwiązanie umożliwia wygenerowanie zespołu przesuniętych w fazie przebiegów zegarowych o częstotliwości $12MHz$, z czego zbocze narastające sygnału *CLK_8* wykorzystywane jest przez odbiornik do czytania danych dokładnie w połowie trwania stanu logicznego. Pozostałe dwa sygnały *CLK_6* i *CLK_10* to przebiegi tworzone przez dekodowanie stanu 6 i 10 licznika C0101 za pośrednictwem bramek logicznych G0102 i G0103. Ponieważ czas trwania poziomu wysokiego tych sygnałów to około $5,2ns$, w przeciwieństwie do *CLK_8* trwającego ponad $41ns$, do ich wydłużenia zostały użyte przerzutniki F0103 oraz F0104. Wejścia danych tych przerzutników zostały na stałe podłączone do stanu wysokiego, natomiast ich wejścia zegarowe sterowane są zboczami narastającymi z bramek G0102 i G0103. Przerzutniki F0103 oraz F0104 resetowane są sygnałem przepełnienia licznika C0101 za pomocą bramki logicznej G0104. Generowane przebiegi zegarowe widoczne są na rysunku (14).



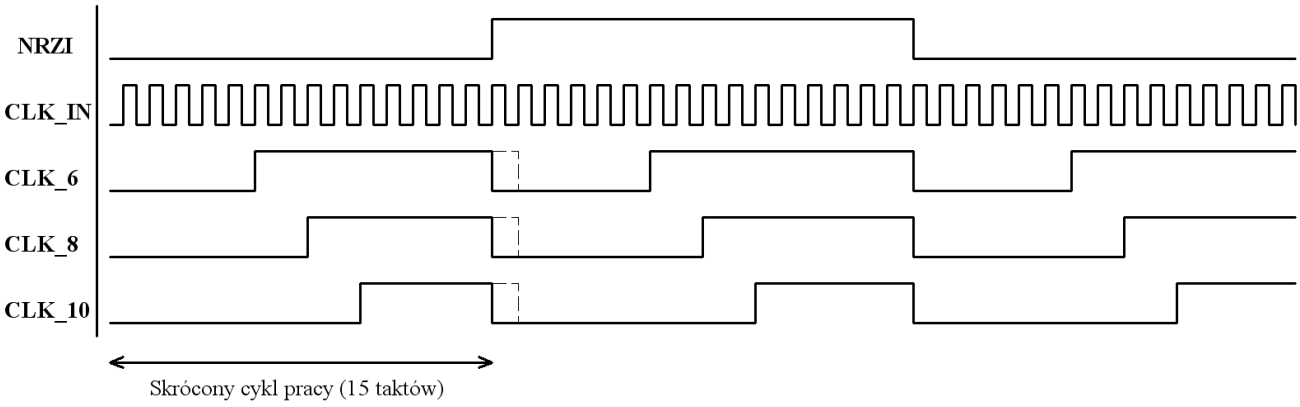
Rysunek 13: Schemat ideowy bloku odzyskiwania przebiegu zegarowego

Dzięki takiemu rozwiązaniu zbocza narastające wszystkich przebiegów zegarowych (*CLK_6*, *CLK_8* i *CLK_10*) pojawiają się z odpowiednim przesunięciem, a zbocza opadające wypadają dokładnie w jednym momencie (przepełnienie licznika lub reset). W urządzeniu wykorzystywane są jedynie zbocza sygnałów zegarowych a nie ich poziomy, więc dla działania całego układu nie jest ważna taka konstrukcja przebiegów taktujących. Pozwala to uniknąć krótkich szpilek wprowadzających zakłócenia, a dodatkowo przebiegi zegarowe w takiej postaci okazały się pomocne i zostały wykorzystane w dalszych blokach funkcjonalnych urządzenia. Wszystkie przebiegi zegarowe są wzmacniane za pomocą globalnych buforów zegarowych B0101...B0103.



Rysunek 14: Wygląd generowanych przebiegów zegarowych

Licznik C0101, oprócz generowania przesuniętych w fazie przebiegów zegarowych pełni jeszcze ważniejszą rolę, a mianowicie zapewnia synchronizację sygnałów zegarowych z danymi podczas odbierania transmisji z komputera. Synchronizacja ta polega na resetowaniu licznika po każdej zmianie stanu logicznego na liniach danych. Za generowanie sygnału resetu dla licznika C0101 odpowiadają przerzutniki F0101 i F0102 oraz inwertery I0101...I0104 i bramka logiczna G0101. Do wejść danych obu przerzutników doprowadzony jest na stałe wysoki stan logiczny, natomiast do wejść *CE* (Count Enable) podłączony jest sygnał *Tx_Dis*. Gdy nadajnik jest włączony (linia *Tx_Dis* w stanie niskim) przerzutniki F0101 i F0102 są zablokowane a przebiegi zegarowe nie są synchronizowane z linią danych. Zapobiega to dostrajaniu się układu do własnych danych. W stanie stacjonarnym, gdy sygnał na linii *Rx_NRZI* nie zmienia się, na wyjściu obu przerzutników panuje stan niski. Dzięki negatorom I0101 oraz I0103 na obu wejściach bramki G0101 występują stany wysokie a na jej wyjściu, będącym jednocześnie sygnałem resetu licznika C0101, panuje stan niski. C0101 nie jest kasowany i odlicza cykle po 16 impulsów generując potrzebne przebiegi zegarowe. Po wystąpieniu zbocza narastającego na linii *Rx_NRZI* przerzutnik F0101 ustawi na swoim wyjściu stan wysoki. Po krótkiej chwili, będącej czasem propagacji inwerterów I0101 i I0102, przerzutnik F0101 zostanie zresetowany. Czas ten w zupełności wystarcza do wykrycia przez bramkę G0101 krótkiej szpilki stanu logicznego niskiego i wygenerowaniu sygnału resetu dla licznika C0101. Dodatkowo za pośrednictwem bramki G0104 resetowane są przerzutniki F0103 i F0104. Przerzutnik F0102 wraz z negatorami I0103, I0104 tworzą drugi taki sam obwód tylko działający na zboczu opadającym linii *Rx_NRZI*.



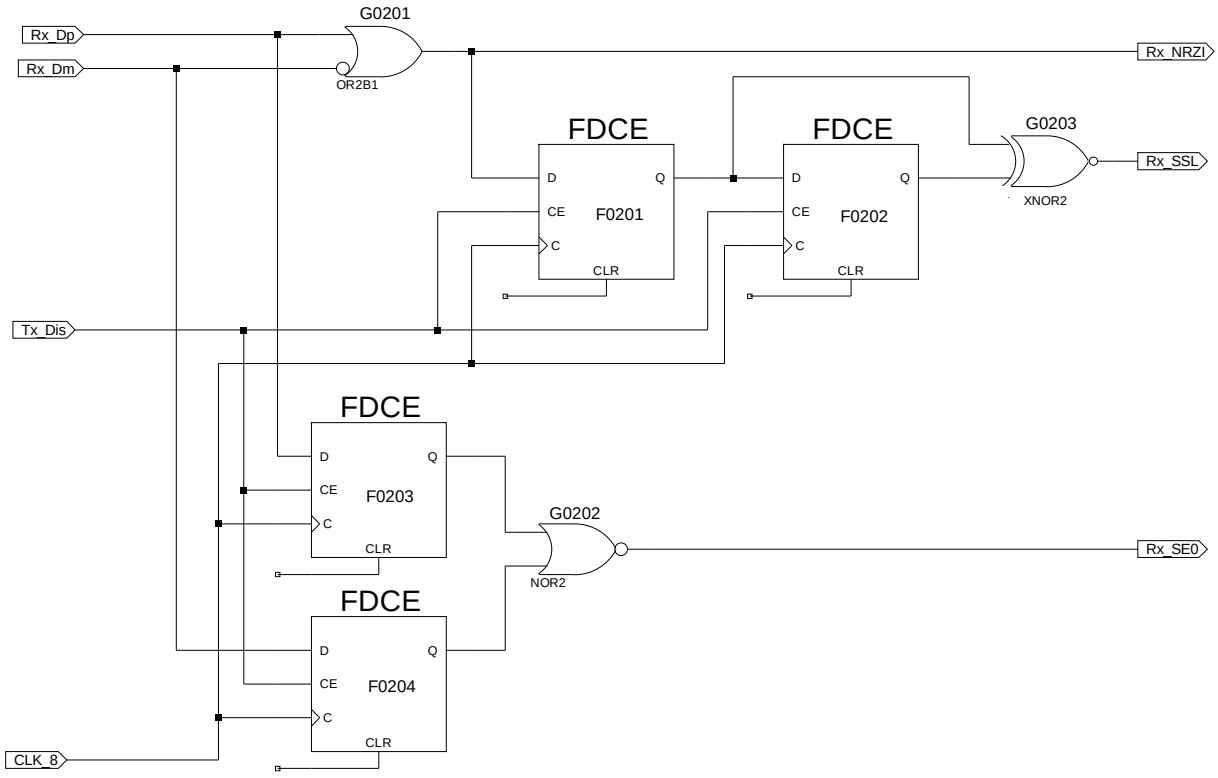
Rysunek 15: Działanie synchronizacji przebiegów zegarowych

Synchronizacja przebiegu zegarowego z sygnałem danych przebiega w następujący sposób: Pierwsze zero (zmiana stanu logicznego na przeciwny) pochodzące z sygnału preambuły (na

początku transmisji) powoduje zresetowanie licznika C0101. Dalej, gdy częstotliwość nadawania sygnału danych ($12MHz$) byłaby dokładnie równa częstotliwości odbiornika (w praktyce tak nigdy nie jest) to sygnał resetu licznika generowany przy każdej zmianie sygnału danych (zerze logicznym) wypadałby dokładnie w punkcie przepełnienia licznika, nie dając żadnego efektu. Gdy natomiast generator zegarowy odbiornika jest troszeczkę wolniejszy od częstotliwości zmian danych to licznik C0101 będzie wcześniej resetowany (przed wystąpieniem przepełnienia, na przykład po doliczeniu do 15). Zapobiega to nawarstwianiu się przesunięcia sygnału danych względem przebiegu zegarowego, w związku z nawet niewielką różnicą ich częstotliwości. Podobny efekt będzie gdy generator zegarowy odbiornika jest szybszy od częstotliwości zmian danych, z tym że licznik C0101 będzie później resetowany (po wystąpieniu przepełnienia, na przykład przy stanie 2). Zasada działania synchronizacji przebiegów zegarowych widoczna jest na rysunku (15).

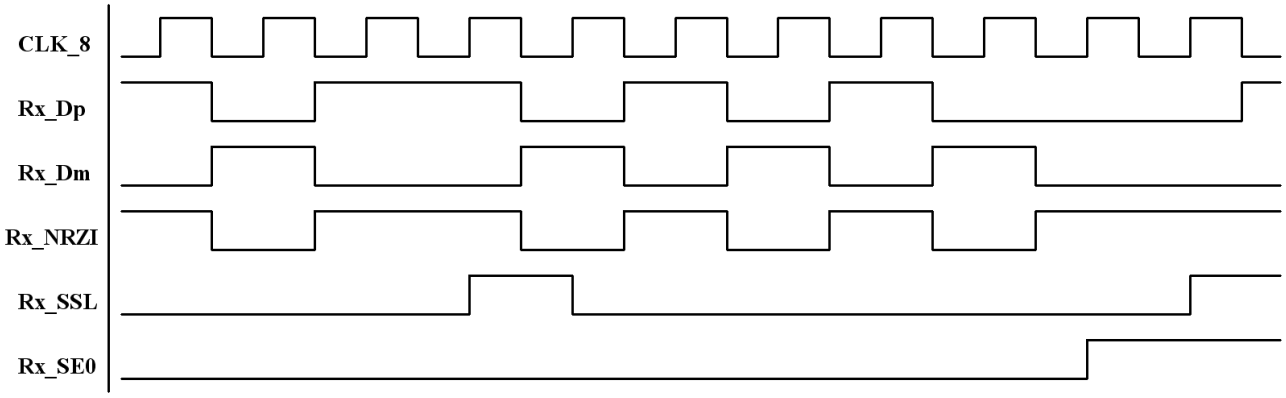
5.2 Blok konwertera sygnałów NRZI na zwykły sygnał logiczny (Rx_Cnv)

Blok konwertera, widoczny na rysunku (16), zajmuje się głównie zamianą danych wysyłanych przez komputer w standardzie NRZI na zwykły sygnał logiczny. Dodatkowo **Rx_Cnv** wykrywa sygnał końca transmisji, będący specjalnym stanem na magistrali USB. Dane nadawane przez komputer różnicowo ($Rx\_Dm = \overline{Rx\_Dp}$, wyjątek stanowi sygnał końca transmisji) są zamieniane na pojedynczy sygnał w standardzie NRZI za pośrednictwem bramki logicznej G0201. Przerzutniki F0201 i F0202 tworzą dwubitowy rejestr przesuwny pracujący na zboczu narastającym sygnału zegarowego *CLK_8*. Do tego rejestru wpisywane są dane z wyjścia bramki G0201. Stany logiczne z wyjść F0201 i F0202 są porównywane przez bramkę logiczną G0203.



Rysunek 16: Blok konwertera sygnałów NRZI na zwykły sygnał logiczny

Ponieważ przerzutniki przechowują zawsze dwa występujące po sobie stany logiczne, za pomocą bramki następuje dekodowanie (zamiana) sygnału NRZI do postaci standardowego sygnału logicznego. Zero logiczne reprezentowane jest w standardzie NRZI za pomocą zmiany stanu logicznego, natomiast jedynka przez brak zmiany stanu logicznego w dwóch bezpośrednio po sobie następujących taktach przebiegu zegarowego. Zwykły przebieg logiczny jest dostępny na wyjściu *Rx_SSL*. Przebiegi czasowe widoczne są na rysunku (17).



Rysunek 17: Wygląd przebiegów czasowych konwertera

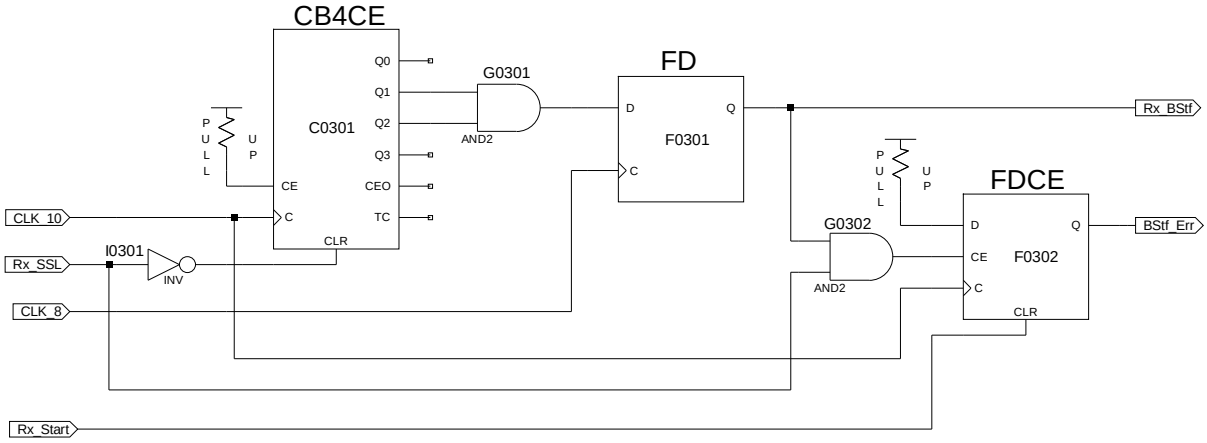
Na magistrali szeregowej, oprócz pojawiania się sekwencji danych, może także wystąpić sygnał SE0 objawiający się niskim stanem logicznym na dwóch liniach: *Rx_Dp* i *Rx_Dm*. Za wykrywanie tego stanu odpowiadają przerzutniki F0203 i F0204 oraz bramka logiczna G0202. Na każdym zboczu narastającym przebiegu *CLK_8* stany logiczne z wejść *Rx_Dp* i *Rx_Dm* są zapamiętywane w przerzutnikach. Bramka logiczna G0202, po wykryciu na swoich wejściach dwóch stanów niskich, wystawia na swoim wyjściu stan wysoki (sygnał *Rx_SE0*). Jest to znak dla pozostałych bloków, że wykryto specjalny stan na magistrali USB. Użycie przerzutników pamiętających stan linii *Rx_Dp* i *Rx_Dm* jest konieczne, ponieważ sama bramka logiczna może generować fałszywe szpilki na linii *Rx_SE0*, wprowadzające w błąd pozostałe bloki logiczne.

Wszystkie przerzutniki pracujące w obrębie bloku konwertera są aktywne gdy na linii *Tx_Dis* panuje stan wysoki (tylko wtedy gdy odbiornik jest włączony).

5.3 Blok wycinania bitów synchronizacyjnych (Rx_BStf)

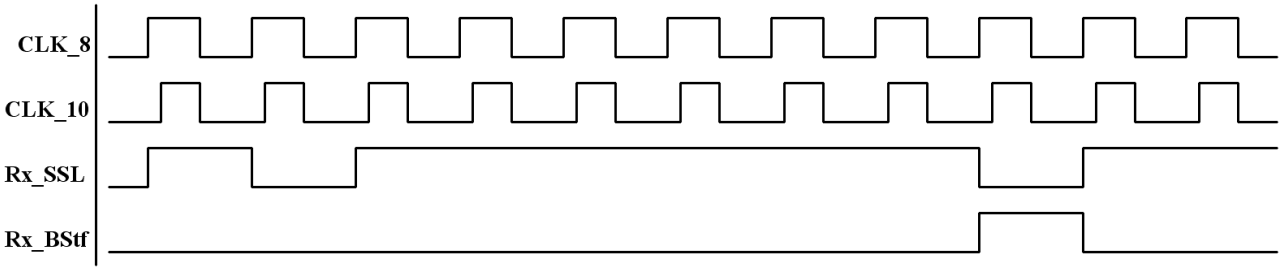
Blok **Rx_BStf** widoczny na rysunku (18) zajmuje się usuwaniem bitów synchronizacyjnych (Bit Stuffing'u). Dodatkowych zer występujących w sygnale danych, a potrzebnych jedynie do synchronizacji układu generowania i odzyskiwania przebiegów zegarowych (**CLK_Rcv_Gen**). Sygnał taktujący *CLK_10* trafia na wejście zegarowe czterobitowego licznika binarnego C0301. Wejście CE (Count Enable) tego licznika jest cały czas w stanie wysokim (jest podciągnięte do +3,3V), zatem licznik jest cały czas aktywny. Do asynchronicznego wejścia resetującego licznika C0301 doprowadzony jest za pomocą inwertera I0301 sygnał danych zwykłego sygnału logicznego (*Rx_SSL*). Ponieważ licznik kasowany jest stanem wysokim na wejściu CLR, każde "zero" danych kasuje jego wartość. Gdy licznik doliczy do wartości 6 oznacza to, że odebrano kolejnych sześć "jedynek", a zatem zgodnie ze specyfikacją uniwersalnej magistrali szeregowej, następna wartość będzie "zerem" i nie będzie ona częścią danych. Stan 6 wystawiony na wyjściach licznika C0301 dekodowany jest za pomocą bramki G0301, która wystawia stan wysoki

na swoim wyjściu, będącym jednocześnie sygnałem danych przerzutnika F0301. Przerzutnik ten przepisze tą wartość na swoje wyjście dopiero w następnym cyklu zegarowym, ponieważ jest on sterowany zboczem narastającym przebiegu zegarowego *CLK_8*, które występuje wcześniej niż zbocze zegara *CLK_10*, na którym został ustawiony stan 6 licznika C0301. W czasie, gdy w kolejnym takcie przychodzi dodatkowe "zero", przerzutnik utrzymuje stan wysoki na swoim wyjściu generując sygnał *Rx_BStf* potrzebny do późniejszego zablokowania odczytu danych na jeden cykl zegarowy.



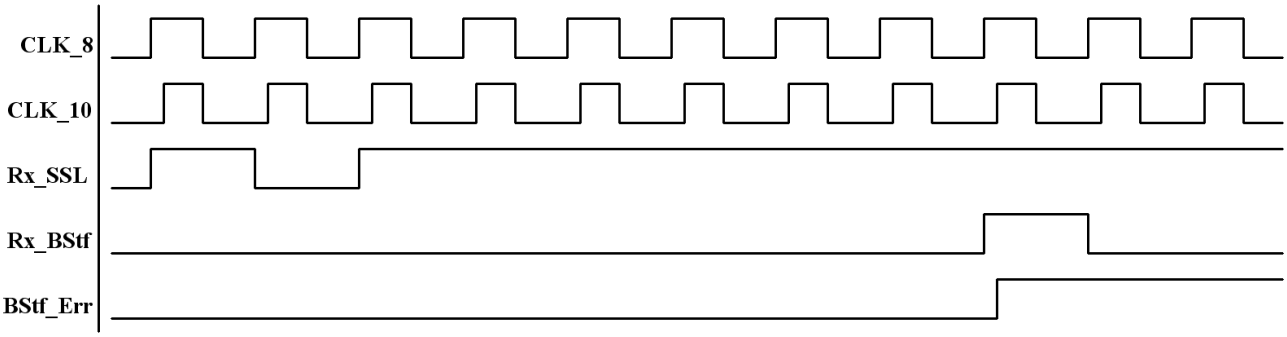
Rysunek 18: Schemat ideowy bloku wycinania bitów synchronizacyjnych

Gdy przerzutnik F0301 ustawi na wyjściu *Rx_BStf* stan wysoki, na linii danych *Rx_SSL* jak już wcześniej wspomniano powinien pojawić się niski stan logiczny. Gdy to nie następuje konieczne jest zgłoszenie błędu do logiki stosu mówiącego o niepoprawnym Bit Stuffing'u. Zadaniem tym obarczony jest przerzutnik F0302 i bramka logiczna G0302. Wejście danych przerzutnika podłączone jest na stałe do wysokiego stanu logicznego. Bramka logiczna G0302 po wykryciu na swoich wejściach dwóch stanów wysokich z linii *Rx_SSL* i *Rx_BStf* ustawia stan wysoki na wejściu CE (Count Enable) przerzutnika F0302 zezwalając tym samym na jego przełączenie na najbliższym zboczu narastającym sygnału zegarowego *CLK_10*. Ustawiony tym sposobem stan wysoki na wyjściu przerzutnika F0302 stanowi jednocześnie sygnał *BStf_Err* niosący informacje dla logiki stosu o błędnie wstawionych bitach synchronizacyjnych. Linia *BStf_Err* będzie w stanie wysokim aż do początku następnej transmisji, gdzie przerzutnik F0302 zostanie zresetowany stanem wysokim sygnału startu *Rx_Start*. Wygląd przebiegów powstających w bloku wycinania bitów synchronizacyjnych w przypadku braku błędów w nadawaniu, przedstawiony jest na rysunku (19).



Rysunek 19: Działanie bloku wycinania Bit Stuffing'u

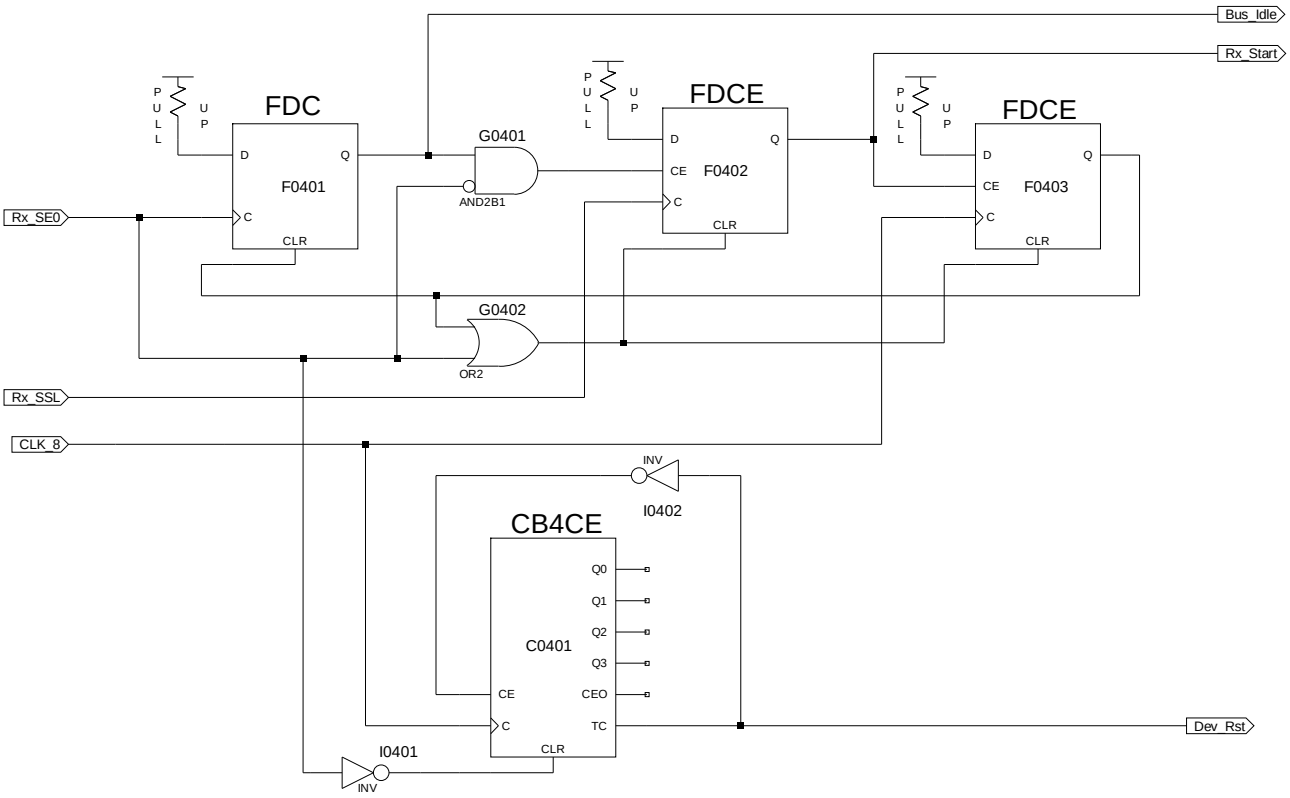
W przypadku gdy na linii danych *Rx_SSL* po wystąpieniu sześciu "jedynek" nie będzie niskiego stanu logicznego to zostanie zgłoszony błąd według zasady widocznej na rysunku (20).



Rysunek 20: Sposób wykrywania błędów Bit Stuffing'u

5.4 Blok monitorowania stanu transmisji (Rx_Idle)

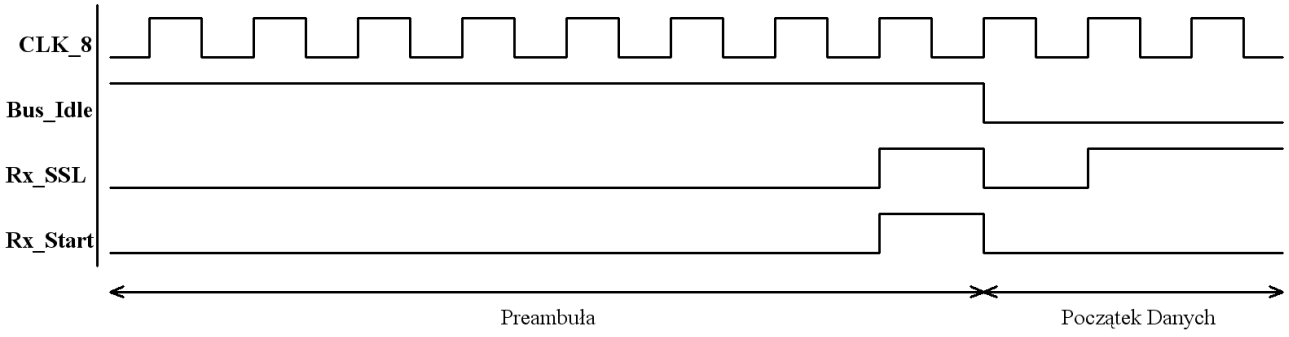
Blok monitorowania stanu transmisji widoczny na rysunku (21) zajmuje się między innymi wycinaniem sygnału synchronizacyjnego (tak zwanej preambuły), generacją sygnału startu *Rx_Start* i wstrzymania *Bus_Idle* a także wykrywaniem stanu resetu na magistrali USB. Na początku wszystkie przerzutniki F0401 - F0403 znajdują się w stanie niskim. Ich wejścia danych zostały na stałe podłączone do wysokiego stanu logicznego.



Rysunek 21: Schemat ideowy bloku monitorowania stanu transmisji

Bezpośrednio po podłączeniu urządzenia do portu USB, komputer wystawia na magistrali sygnał resetu[12], który polega na utrzymywaniu sygnału SE0 (obie linie danych w stanie niskim) przez czas co najmniej 10ms. Za wykrycie tego stanu odpowiada czterobitowy licznik binarny C0401 oraz negator I0401. Po zliczeniu 15 cykli zegarowych w których linia *Rx_SE0* (oznaczająca wykryty sygnał SE0) utrzymuje stan wysoki, zostanie ustawiony wysoki stan logiczny na wyjściu *Dev_Rst* aż do końca trwania sygnału *Rx_SE0*. *Dev_Rst* posłuży w logice stosu to ustawienia stanów początkowych w rejestrach urządzenia. Licznik C0401 jest kasowany stanem niskim na linii *Rx_SE0* a więc reset urządzenia zostanie wykryty tylko na początku po włączeniu do portu USB(Żadna inna sytuacja na magistrali nie ustawia na tak długi czas sygnału SE0). W celu zapewnienia jednorazowego sygnału *Dev_Rst* (licznik C0401 jest w stanie przepełnić się wielokrotnie w czasie trwania sygnału SE0 zaraz po podłączeniu urządzenia do magistrali) został użyty inwerter I0402. Na początku na wyjściu tego inwertera panuje stan wysoki (ponieważ licznik ma wszystkie wyjścia w stanie niskim), który jest jednocześnie sygnałem CE licznika C0401 umożliwiając zliczenie kolejnych taktów zegara. Po wystąpieniu stanu wysokiego na linii *Dev_Rst* negator I0402 wystawi stan niski na wejściu CE licznika C0401, uniemożliwiając tym samym wystąpienia kolejnego sygnału *Dev_Rst* aż do momentu stanu niskiego na *Rx_SE0*

Pojawiające się, zaraz po włączeniu urządzenia, narastające zbocze na linii *Rx_SE0* ustawi stan wysoki na wyjściu przerzutnika F0401. Wyjście to stanowi jednocześnie linię *Bus_Idle*, która razem z informacją pochodzącą z bloku **Rx_BStf** stanowi sygnał informujący stos kiedy ma odczytywać dane z układu Transceiver'a. Wysoki stan logiczny na linii *Bus_Idle* oznacza że odbierane dane są preambułą lub magistrala USB nie jest używana i jest na niej ustawiony stabilny stan logiczny. Odpowiada to ciągłemu stanowi wysokiemu na linii danych (*Rx_SSL*). Gdy sygnał *Rx_SE0* powróci do stanu niskiego bramka G0401 mając na swoich wyjściach stan wysoki z przerzutnika F0401 i stan niski z *Rx_SE0* ustawi stan wysoki na wejściu CE przerzutnika F0402. Na jego wejście zegarowe podawane są dane w postaci zwykłego sygnału logicznego (*Rx_SSL*). Jak już wcześniej wspomniano linia *Rx_SSL* jest cały czas w stanie wysokim gdy nie są przesyłane dane po magistrali USB, natomiast na początku preambuły linia ta przechodzi w stan niski. Ponieważ przerzutnik F0402 przepisuje stan logiczny na narastającym zboczu swojego sygnału zegarowego, narastające napięcie na linii *Rx_SSL* (pojawiające się na końcu preambuły, która jest sygnałem postaci siedmiu "zer" logicznych zakończonych "jedynką") ustawi stan wysoki na wyjściu F0402, będącym jednocześnie sygnałem zezwalającym CE kolejnego przerzutnika (F0403) i sygnałem *Rx_Start*.



Rysunek 22: Przebiegi generowane na początku każdej transmisji

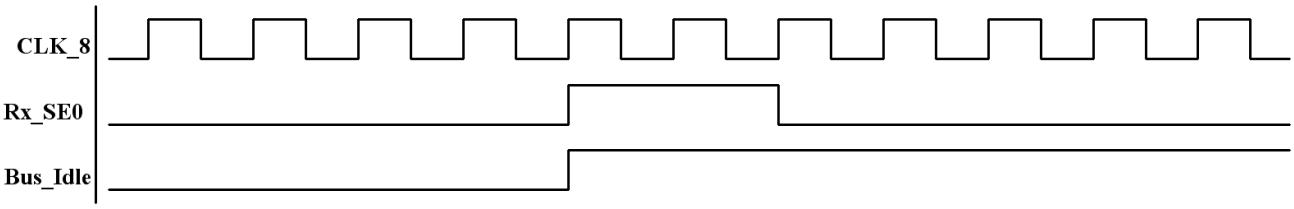
Wejście zegarowe F0403 podłączone jest do przebiegu taktującego *CLK_8*, co w efekcie spra-

[12]Universal Serial Bus Specification, rev. 2.0, s. 153

wi że na kolejnym zboczu *CLK_8*, na wyjściu F0403 zostanie wygenerowana szpilka napięcia. Czas trwania tego impulsu jest uzależniony od czasu propagacji przerzutnika F0403 oraz bramki G0402. Po jego pojawieniu się przerzutnik F0401 zostanie zresetowany i ustawi niski stan na linii *Bus_Idle* umożliwiając tym samym odczyt danych przez logikę stosu. Bramka G0402 jest odpowiedzialna za resetowanie przerzutników F0402 i F0403 zarówno szpilką napięcia z wyjścia F0403 jak i stanem wysokim na linii *Rx_SE0*. Pozwala to uniknąć przypadkowych stanów logicznych zapamiętanych w przerzutnikach gdy urządzenie nie jest podłączone do magistrali a stany na liniach danych są nieokreślone (wejścia danych D+ i D- "wiszą" w powietrzu). Przebiegi występujące na początku każdej transmisji zostały przedstawione na rysunku (22)

Przerzutnik F0402 działa na zboczu narastającym pojawiającym się na ostatnim bicie preambuły. Wtedy też sygnał *Rx_Start* ustawiany jest w stan wysoki i trwa aż do pojawienia się pierwszego bitu danych. Jest to spowodowane tym, że dane odczytywane są przez blok konwertera (*Rx_Cnv*) na zboczu narastającym tego samego przebiegu zegarowego (*CLK_8*). Zanim logika generująca sygnał *Rx_SSL* zadziała, wystawiając wysoki stan logiczny na tym wyjściu, sygnał zegarowy *CLK_8* będzie już w stanie wysokim i nie spowoduje przełączenia przerzutnika F0403. Przełączenie będzie możliwe dopiero na następnym zboczu narastającym przebiegu zegarowego.

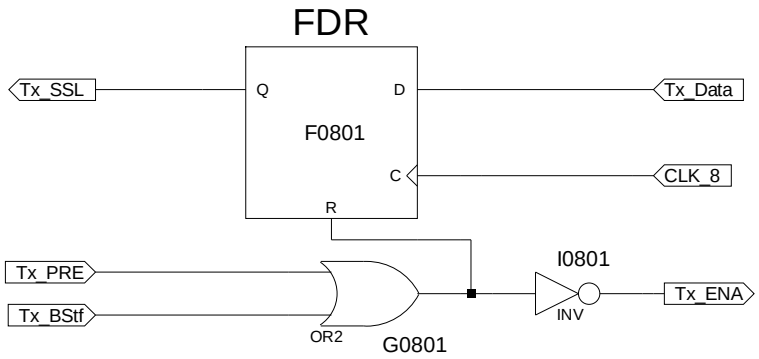
Na rysunku (23) przedstawiono sposób generacji sygnału *Bus_Idle* na końcu odbierania pakietu danych z komputera.



Rysunek 23: Tworzenie sygnału *Bus_Idle* na końcu transmisji

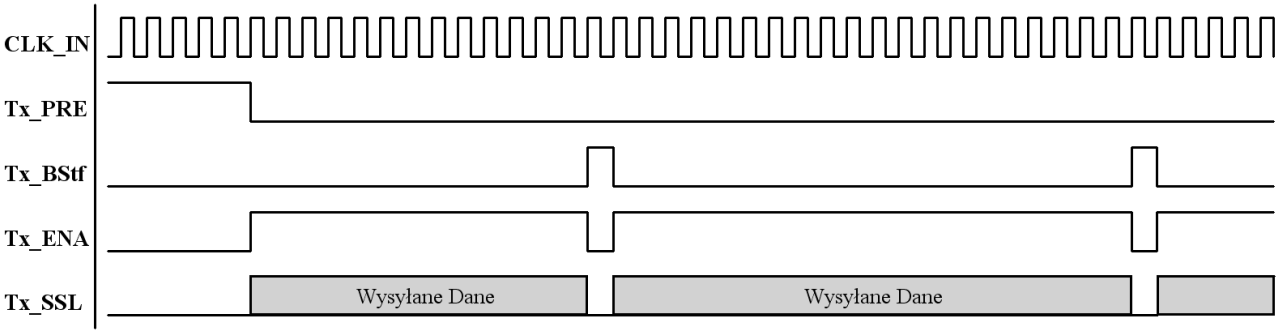
5.5 Układ wejściowy nadajnika (Tx_In)

Blok wejściowy nadajnika widoczny jest na rysunku (24). Pośredniczy on między logiką stosu a obwodami nadajnika. Dodatkowo blok ten pełni rolę modułu wykonawczego dla układu wstawiania bitów synchronizacyjnych (**Tx_BStf**). Po uruchomieniu nadajnika (punkt 4.2.2) stos wystawia pierwszy bit danych na wejście przerzutnika F0801 (sygnał *Tx_Data*), wartość ta będzie przepisywana na jego wyjście na każdym zboczu narastającym przebiegu zegarowego *CLK_8* pod warunkiem, że na R (synchroniczny reset) przerzutnika F0801 występuje niski stan logiczny. W przeciwnym wypadku (gdy $R = 1$) wyjście przerzutnika będzie cały czas w stanie niskim niezależnie od sygnału danych. Za generację sygnału resetu dla F0801 odpowiada bramka logiczna G0801. Na jej wejścia podawane są sygnały *Tx_PRE* i *Tx_BStf*. Linia *Tx_PRE* jest ustawiana w stan wysoki podczas nadawania preambuły, natomiast linia *Tx_BStf* przyjmuje wysoki stan logiczny gdy na linii *Tx_SSL* zostało zliczonych kolejno sześć "jedynek" logicznych. Dzięki bramce G0801 sterującej wejściem resetu przerzutnika F0801 blokowany jest przepływ danych na czas nadawania preambuły i podczas konieczności wstawiania bitów synchronizacyjnych.



Rysunek 24: Schemat ideowy układu wejściowego nadajnika

Aby w czasie tych czynności kolejne bity danych nie były wystawiane przez stos, jego bufor wyjściowy uaktywniany jest sygnałem *Tx_ENA* uzyskiwanym z zanegowanego sygnału resetu przerzutnika F0801 za pomocą inwertera I0801. Podczas trwania preambuły (wysoki stan na *Tx_PRE*) lub podczas wstawiania bitów synchronizacyjnych (wysoki stan na *Tx_BStf*) zostanie wystawiony stan niski na linii *Tx_ENA*. Podsumowując sygnał *Tx_ENA* stanowi informację o tym że poprzednia dana została wysłana i należy wystawić kolejny bit na wejście *Tx_Data* na najbliższym zboczu przebiegu zegarowego. Na rysunku (25) przedstawiono przebiegi sygnałów logicznych obecnych w układzie wejściowym nadajnika. Przedstawiają one zachowanie układu podczas wstawiania bitów synchronizacyjnych i generowania preambuły.

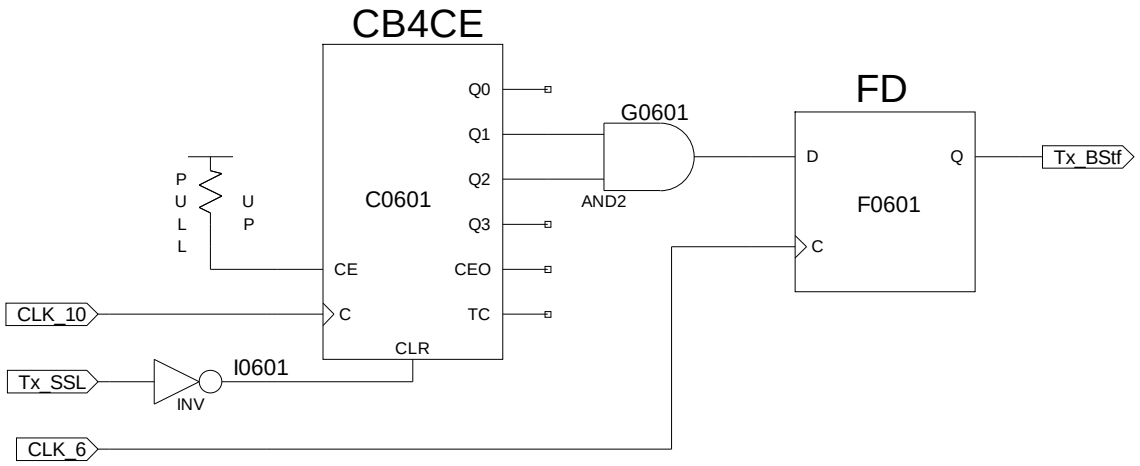


Rysunek 25: Sygnały logiczne obecne w układzie wejściowym nadajnika

5.6 Blok wstawiania bitów synchronizacyjnych (Tx_BStf)

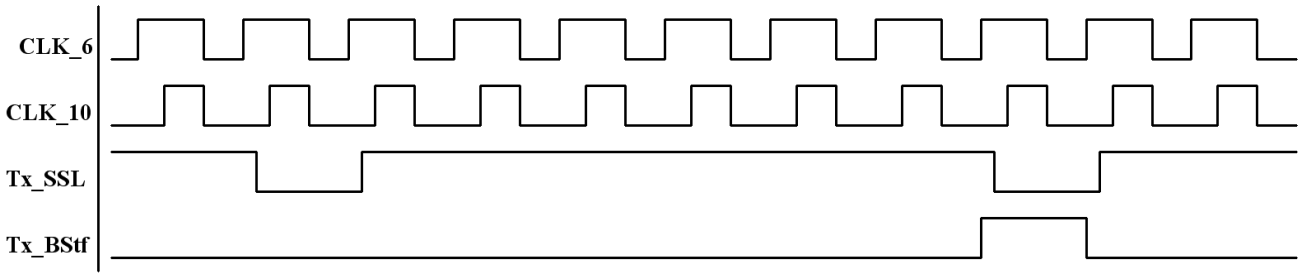
Układ wstawiania bitów synchronizacyjnych (Bit Stuffing'u) przedstawiony jest na rysunku (26). Jego zadaniem jest monitorowanie danych wysyłanych na magistralę USB i po wykryciu sześciu "jedynek" logicznych dalszy odczyt danych ze stosu jest wstrzymany na jeden takt przebiegu zegarowego umożliwiając wysłanie zera synchronizacyjnego.

Do wejścia zegarowego C0601 podłączony jest sygnał zegarowy *CLK_10*. Wejście CE licznika podłączone jest na stałe do wysokiego stanu logicznego a zatem będzie on zwiększał wartość liczbową na wyjściu na każdym zboczu narastającym sygnału *CLK_10* pod warunkiem, że na jego wejściu resetu CLR będzie stan niski. Wejście CLR licznika C0601 ustawiane jest za pomocą inwertera I0601 na którego podawany jest sygnał danych *Tx_SSL*. W efekcie każde zero



Rysunek 26: Schemat ideowy bloku wstawiania Bit Stuffing'u

logiczne z sygnału danych powoduje reset licznika a zatem gdy bramka G0601 wykryje na swoich wejściach wysokie stany logiczne(co nastąpi po zliczeniu sześciu "jedynek") wystawi sygnał wysoki na wejście przerzutnika F0601. Ponieważ wejście zegarowe tego przerzutnika podłączone jest do sygnału zegarowego *CLK_6* (występującego wcześniej niż *CLK_10*) przerzutnik F0601 przepisze stan wysoki na swoje wyjście dopiero w następnym cyklu zegarowym. Wygenerowany tym sposobem sygnał *Tx_BStf* niesie informacje dla bloku **Tx_IN** o konieczności zablokowania pobierania danych na jeden cykl zegarowy i wystawieniu zera logicznego na linię danych. W następnym cyklu zegarowym linia *Tx_BStf* powróci do stanu niskiego ponieważ licznik C0601 zostanie zresetowany wstawionym zerem synchronizacyjnym a bramka G0601 wystawi stan niski na swoim wyjściu. Przebiegi generowane w tym bloku widoczne są na rysunku (27).

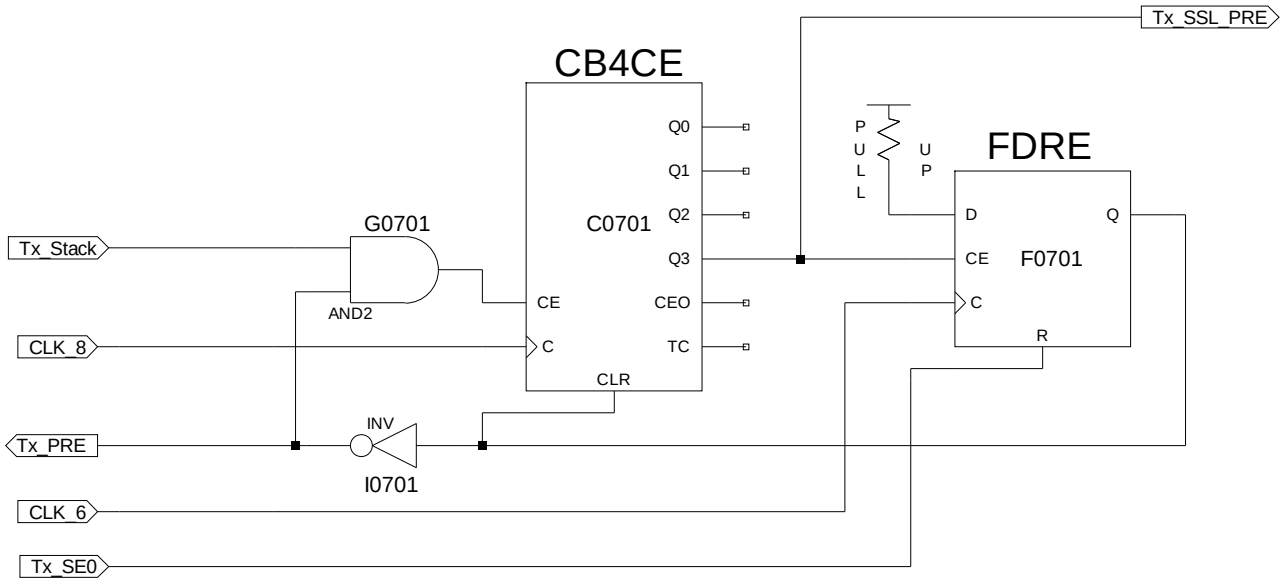


Rysunek 27: Działanie bloku wstawiania Bit Stuffing'u

5.7 Blok generacji preambuły (Tx_PRE)

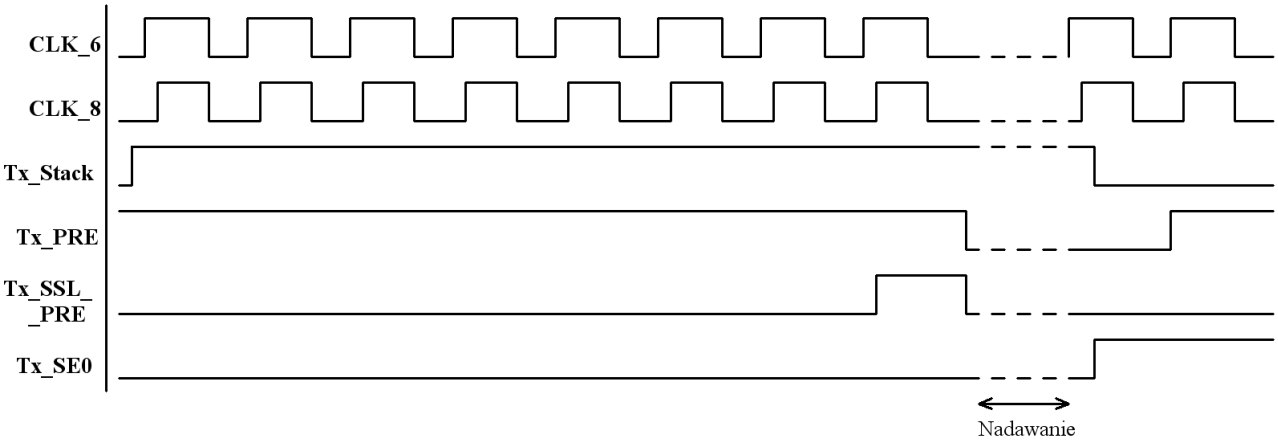
Blok generacji preambuły, widoczny na rysunku (28), ma za zadanie rozpocząć nadawanie bajtu synchronizacji po wykryciu stanu wysokiego na linii *Tx_Stack*. Na początku przerzutnik F0701 i licznik C0701 są zresetowane, a zatem zero logiczne z wyjścia F0701 za pośrednictwem inwertera I0701 ustawia stan wysoki na linii *Tx_PRE* i wejściu bramki logicznej G0701. Sygnał *Tx_PRE* wykorzystywany jest w multiplekserze bloku konwertera **Tx_Cnv** do wyboru czy mają być nadawane bity preambuły czy bity danych.

Gdy stos rozpoczyna nadawanie ustawiając stan wysoki na linii *Tx_Stack*, za pośrednictwem bramki G0701 zostanie ustawiony stan wysoki na wejściu CE (Count Enable) licznika binarnego



Rysunek 28: Schemat ideowy bloku generacji preambuły

C0701. Kolejne zbocza narastające sygnału *CLK_8* będą zwiększały wartość na wyjściu licznika i gdy wystąpi wysoki stan logiczny na wyjściu Q3 (który jest jednocześnie sygnałem danych preambuły *Tx_SSL_PRE*) zostanie odblokowany przerzutnik F0701. Ponieważ wejście zegarowe tego przerzutnika sterowane jest sygnałem *CLK_6*, przychodzącym wcześniej niż *CLK_8*, na wyjście przerzutnika (w następnym cyklu zegarowym) zostanie wpisany stan wysoki, który zresetuje C0701 i za pośrednictwem inwertera I0701 ustawi stan niski na linii *Tx_PRE* (koniec preambuły) oraz bramce G0701. Stan niski na wyjściu bramki G0701 zablokuje licznik C0701. Po wystąpieniu stanu wysokiego na *Tx_SE0* (koniec transmisji) na najbliższym zboczu zegarowym *CLK_6* przerzutnik F0701 wystawi stan niski na swoim wyjściu zapewniając powrót układu do warunków początkowych (oczekiwanie na wysyłanie kolejnego pakietu)



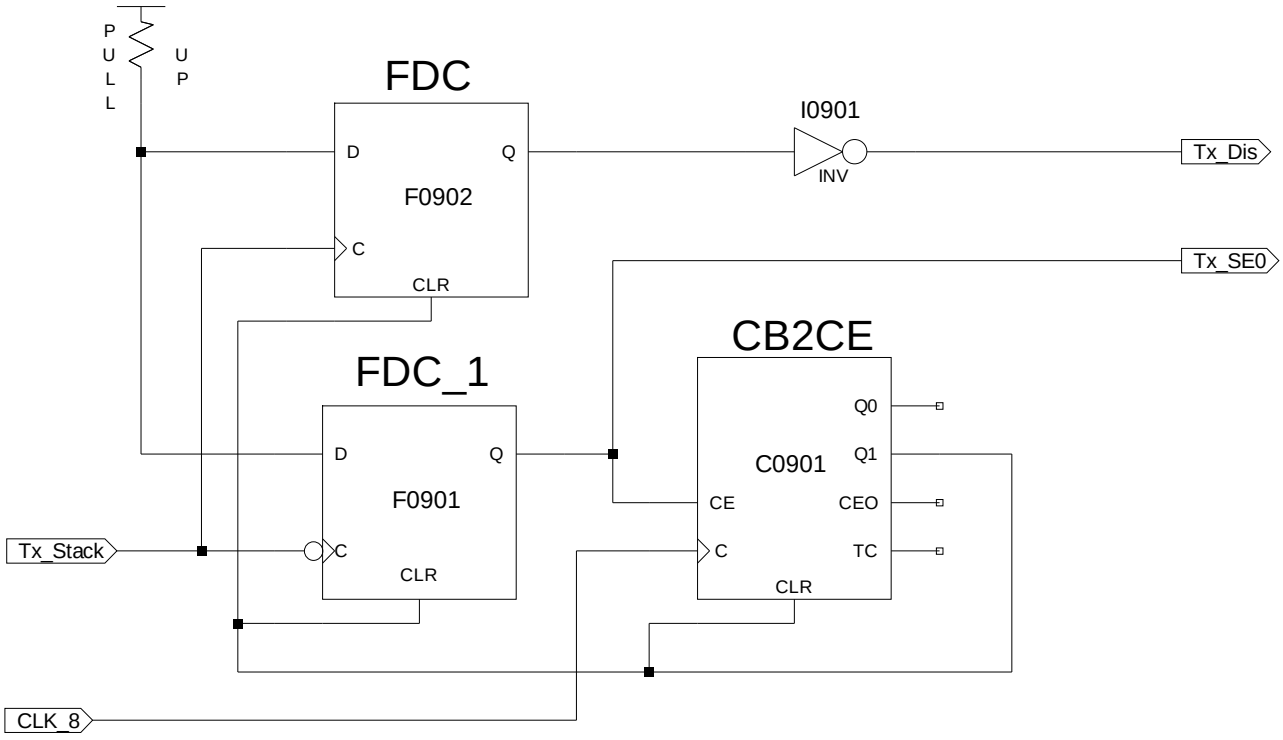
Rysunek 29: Przebiegi w bloku generacji preambuły

W efekcie za pomocą C0701 wystawiony zostanie sygnał preambuły składający się z siedmiu "zer" zakończonych "jedynką" logiczną, po czym zostanie wystawione zezwolenie na wysyła-

nie danych. Po zakończeniu transmisji układ będzie oczekiwał na kolejne wystawienie stanu wysokiego na linii *Tx_Stack*. Na rysunku (29) przedstawiono zestaw przebiegów logicznych tworzonych w bloku generacji preambuły na początku i końcu każdej transmisji do komputera.

5.8 Blok generacji sygnału końca transmisji (Tx_EOP)

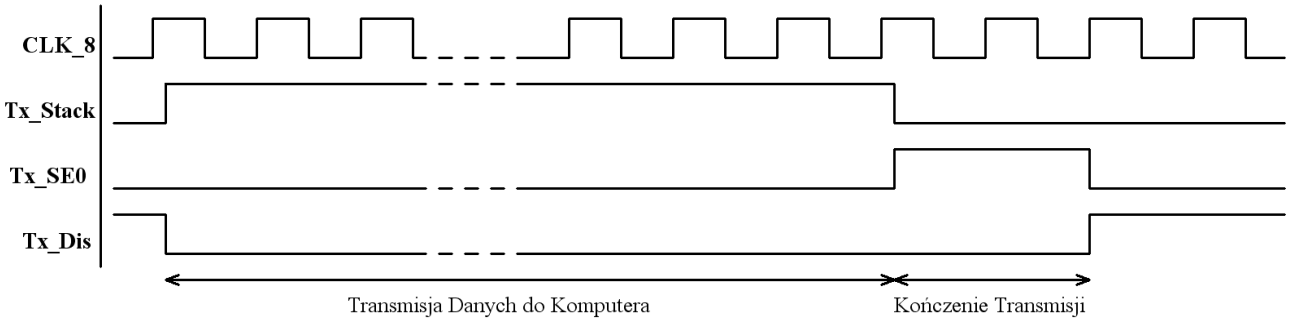
Blok generacji sygnału końca transmisji, widoczny na rysunku (30), ma za zadanie wykryć koniec sygnału *Tx_Stack* (sygnał chęci nadawania ze strony stosu) i wystawić na dwa cykle zegarowe stan *Tx_SE0* informujący układ konwertera **Tx_Cnv**, że na magistralę USB ma być wystawiony sygnał końca transmisji.



Rysunek 30: Schemat ideowy bloku generacji sygnału końca transmisji

Przerzutnik F0901, którego wejście danych podłączone jest na stałe do wysokiego stanu logicznego, przepisuje ten stan na swoje wyjście na zboczu opadającym sygnału *Tx_Stack* ustawiając tym samym linię *Tx_SE0* w stan wysoki. Wyjście F0901 podawane jest jednocześnie na dwubitowy licznik binarny C0901 jako sygnał CE. Kolejne zbocza sygnału zegarowego *CLK_8* podłączonego na wejście zegarowe licznika zwiększają wartość na wyjściu C0901. Gdy na Q1 pojawi się stan wysoki (wartość 2 w liczniku C0901) będzie on powodem resetu przerzutnika F0901 i licznika C0901. Linia *Tx_SE0* powróci zatem do stanu niskiego. W efekcie blok **Tx_EOP** wystawi sygnał *Tx_SE0* na dwa takty zegarowe, co jest zgodne ze specyfikacją USB.

Blok generacji sygnału końca transmisji został obarczony dodatkowo zadaniem sterowania dwukierunkowymi buforami wejściowymi Transceiver'a a co za tym idzie sterowanie kierunkiem przepływu danych (nadawanie lub odbieranie). Odbywa się to dzięki generacji sygnału *Tx_Dis*. Na narastającym zboczu sygnału *Tx_Stack* przerzutnik F0902, którego wejście danych podłączone jest na stałe do wysokiego stanu logicznego, przepisze stan wysoki na swoje wyjście, co za

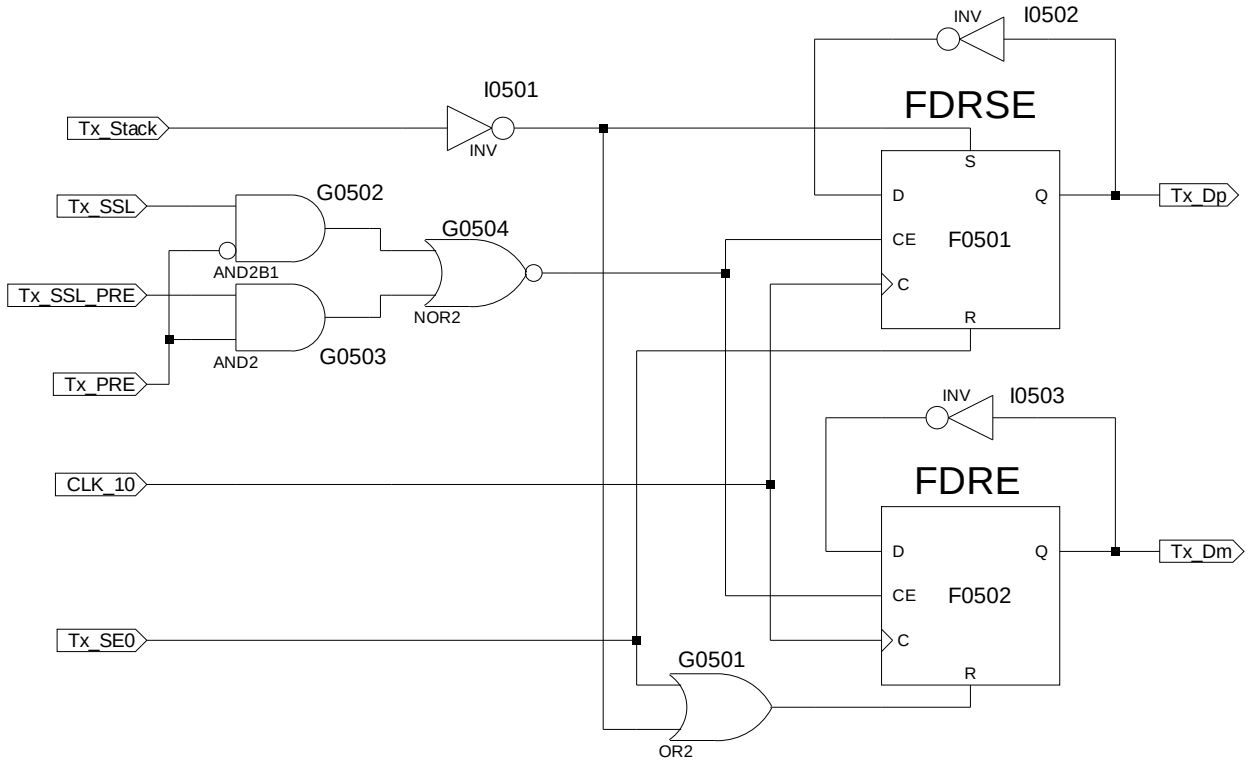


Rysunek 31: Przebiegi logiczne bloku generacji sygnału końca transmisji

pośrednictwem inwertera I0901 spowoduje pojawienie się stanu niskiego na linii *Tx_Dis* i uruchomienie nadajnika. Sygnał *Tx_Dis* zostanie ustawiony w stan wysoki dopiero po zakończeniu nadawania sygnału końca transmisji. Widok przebiegów logicznych na początku i końcu fazy nadawania sygnału przedstawia rysunek (31)

5.9 Blok konwertera zwykłego sygnału logicznego na NRZI (Tx_Cnv)

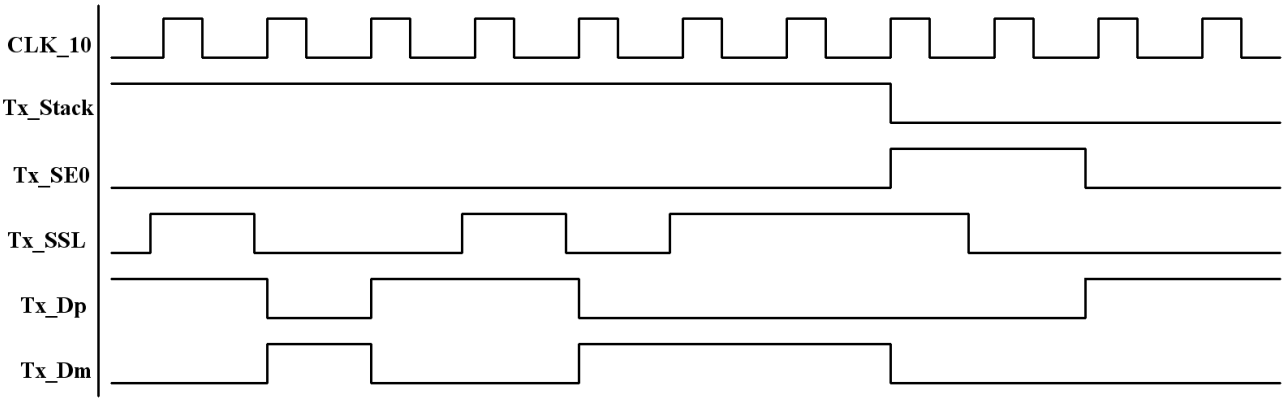
Blok konwertera sygnałów, którego schemat znajduje się na rysunku (32) ma za zadanie zamieniać sygnał logiczny pochodzący z bloku preambuły lub ze stosu, na różnicowy sygnał w standardzie NRZI. Bramki G0502, G0503 i G0504 tworzą multiplekser z negacją sygnału na wyjściu, sterowany stanem logicznym na linii *Tx_PRE*.



Rysunek 32: Schemat ideowy bloku konwertera zwykłego sygnału logicznego na NRZI

Wysoki stan logiczny na *Tx_PRE* umożliwia przekaz sygnału z bloku preambuły na wyjście bramki G0504, natomiast stan niski na *Tx_PRE* umożliwia transmisję sygnału danych na wyjście G0504. Sygnał z tej bramki logicznej podawany jest na wejścia CE przerzutników F0501 i F0502 natomiast na ich wejścia zegarowe podawany jest przebieg zegarowy *CLK_10*. Inwertery I0502 oraz I0503 umożliwiają zmianę stanów logicznych w przerzutnikach na przeciwne po wystąpieniu narastającego zbocza zegarowego na *CLK_10* o ile ich sygnały CE są w stanie wysokim.

W stanie stacjonarnym, linie *Tx_Stack* i *Tx_SE0* są w stanie niskim. Za pośrednictwem inwertera I0501 przerzutnik F0501 zostanie ustawiony w stan wysoki, natomiast F0502 dodatkowo za pomocą bramki G0501 zostanie ustawiony w stan niski. Podczas transmisji danych, kiedy linia *Tx_Stack* jest w stanie wysokim, a linia *Tx_SE0* jest w stanie niskim, wyjścia przerzutników zależą tylko od danych wystawianych na wyjściu bramki G0504. Jedynka logiczna na zanegowanym wyjściu bramki G0504 powoduje wystawienie stanu niskiego na wejściach CE przerzutników F0501 i F0502 a tym samym zablokowanie zmiany sygnału na przeciwny na następnym zboczu narastającym sygnału zegarowego (jedynka logiczna w standardzie NRZI). Natomiast zero logiczne wywoła zmianę stanu wyjść przerzutników na przeciwne (zero logiczne w standardzie NRZI). Gdy stos zakończy nadawanie i linia *Tx_Stack* przyjmie wartość 0 zostanie ustawiony stan wysoki na linii *Tx_SE0*. W efekcie przerzutnik F0501 zostanie zresetowany podobnie jak (za pośrednictwem G0501) F0502 a na liniach *Tx_Dp* i *Tx_Dm* pojawią się niskie stany logiczne na czas trwania sygnału *Tx_SE0*. Po tym fakcie układ konwertera powróci do stanu początkowego oczekując na kolejną transmisję. Na rysunku (33) przedstawiono przebiegi logiczne układu konwertera podczas nadawania danych i sygnału końca transmisji.



Rysunek 33: Działanie bloku konwertera zwykłego sygnału logicznego na NRZI

6 Stos USB, opis ogólny

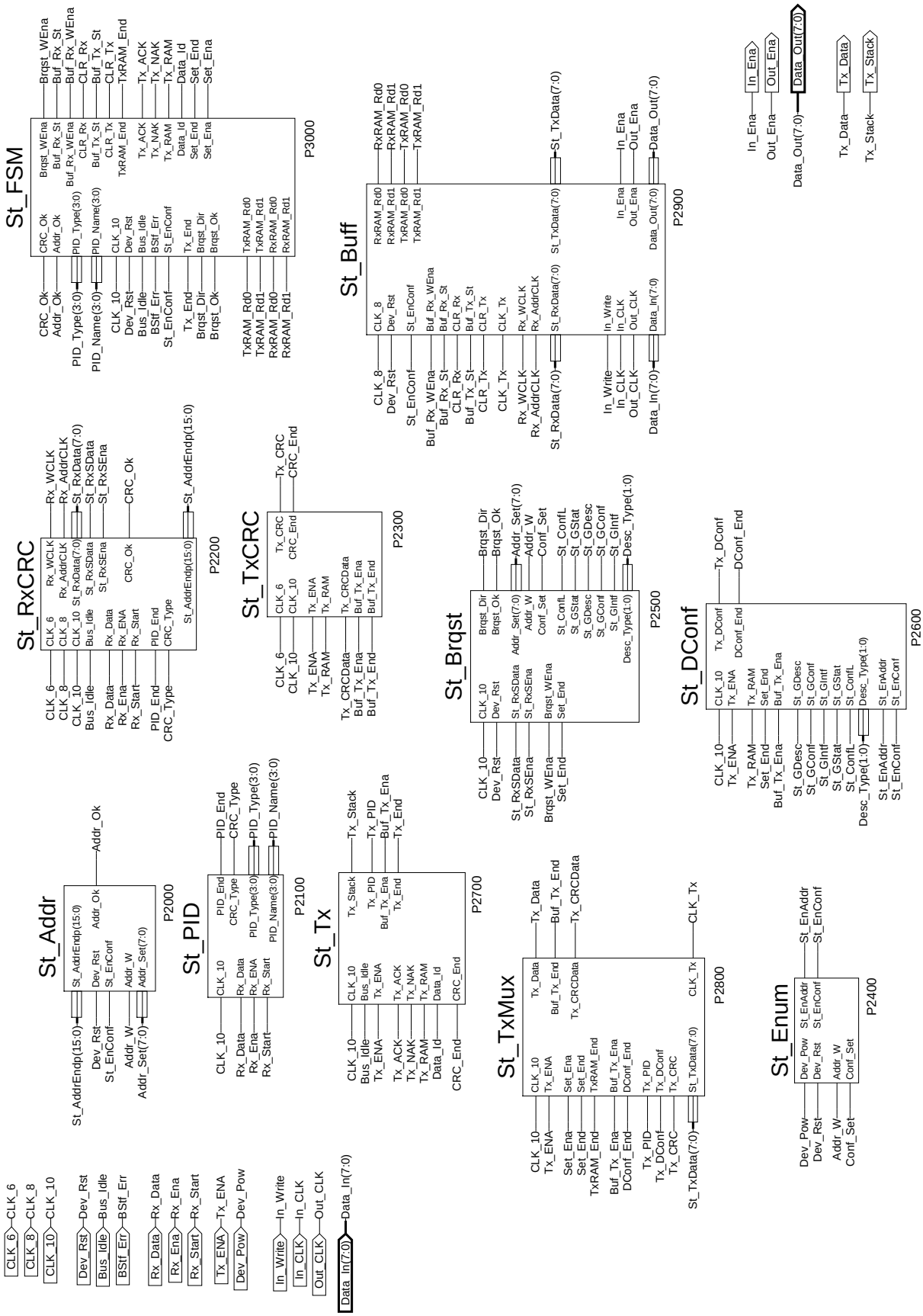
6.1 Wstęp

Zadaniem stosu jest interpretacja oraz generacja pakietów odbieranych i nadawanych poprzez uniwersalną magistralę szeregową USB. Dzięki rozpoznawaniu protokołów USB stos umożliwia zewnętrznemu urządzeniu wymianę danych z komputerem. Dodatkową, choć konieczną i dość rozbudowaną, funkcją pełnioną przez stos jest wstępna konfiguracja umożliwiająca komputerowi poprawne rozpoznanie i obsługę interfejsu. Analiza protokołów wykonywana przez stos polega na rozpoznaniu typu odbieranego pakietu, sprawdzeniu zgodności adresu interfejsu z adresem zawartym w pakiecie (oprócz pakietów danych) oraz kontroli poprawności cyklicznego kodu nadmiarowego CRC. Dalsze działania podejmowane przez stos, takie jak transmisja danych do/z urządzenia zewnętrznego, obsługa konfiguracji interfejsu, są uzależnione od treści generowanych przez komputer pakietów typu 'Token', pełniących niejako rolę rozkazów sterujących transmisją. Stos pełni rolę pośrednika pomiędzy urządzeniem zewnętrznym, wydostając bądź obudowując przesyłane dane w odpowiednią strukturę protokołów, a częścią sprzętową (Transceiver), która nadaje bądź odbiera dowolne pakiety bez analizy ich treści.

6.2 Schemat blokowy stosu USB

Na rysunku (34) przedstawiono schemat blokowy stosu. Sygnały wejściowe i wyjściowe układu można podzielić na dwie grupy - służące komunikacji z Transceiver-em (sygnały informujące o stanie transmisji i interfejsu - *Dev_Rst*, *Dev_Pow*, *Bus_Idle*, *BStf_Err*, *Rx_Ena*, *Rx_Start*, *Tx_ENA*, *Tx_Stack*, sygnały zegarowe *CLK_6*, *CLK_8*, *CLK_10* oraz linie danych *Rx_Data*, *Tx_Data*), a także sygnały umożliwiające komunikację z urządzeniem zewnętrznym (sterujące *In_Write*, *In_Ena*, *Out_Ena*, zegarowe *In_CLK*, *Out_CLK* oraz magistrale danych *Data_In(7:0)*, *Data_Out(7:0)*).

Dane odbierane przez Transceiver są przesyłane w pierwszej kolejności do bloku zajmującego się analizą nagłówka pakietu **St_PID**. Blok ten odcina pierwszy bajt pakietu zawierający nagłówek, zaś pozostała część pakietu zostaje równolegle przesłana do bloku sprawdzającego adres **St_Addr**, bloku kontrolującego poprawność cyklicznego kodu nadmiarowego (zwanego inaczej sumą kontrolną) **St_RxCRC** i, jeżeli pakiet zawiera dane, do bufora wyjściowego znajdującego sięw bloku **St_Buff** lub do bloku interpretującego rozkazy konfiguracyjne **St_Brqst**. Konstrukcja pakietu przeznaczonego do nadania rozpoczyna się od generacji odpowiedniego nagłówka w bloku **St_Tx**. Następnie do nagłówka, poprzez blok **St_TxMux** zajmujący się składaniem części pakietu w całość, dołączone zostają dane z bufora wejściowego interfejsu (**St_Buff**) bądź, w fazie konfiguracji, odpowiednie deskryptory z bloku **St_DConf**. Równolegle dane te zostają przesłane do bloku obliczającego cykliczny kod nadmiarowy **St_TxCRC**, który jest następnie dołączany na końcu konstruowanego pakietu. Blok **St_Enum** przechowuje informację o postępie wstępnej konfiguracji interfejsu (tzw. enumeracji USB). Zarząd nad całością działań podejmowanych przez stos pełni maszyna stanów skończonych **St_FSM**. Dokładny opis działania poszczególnych bloków zamieszczono w rozdziale 7

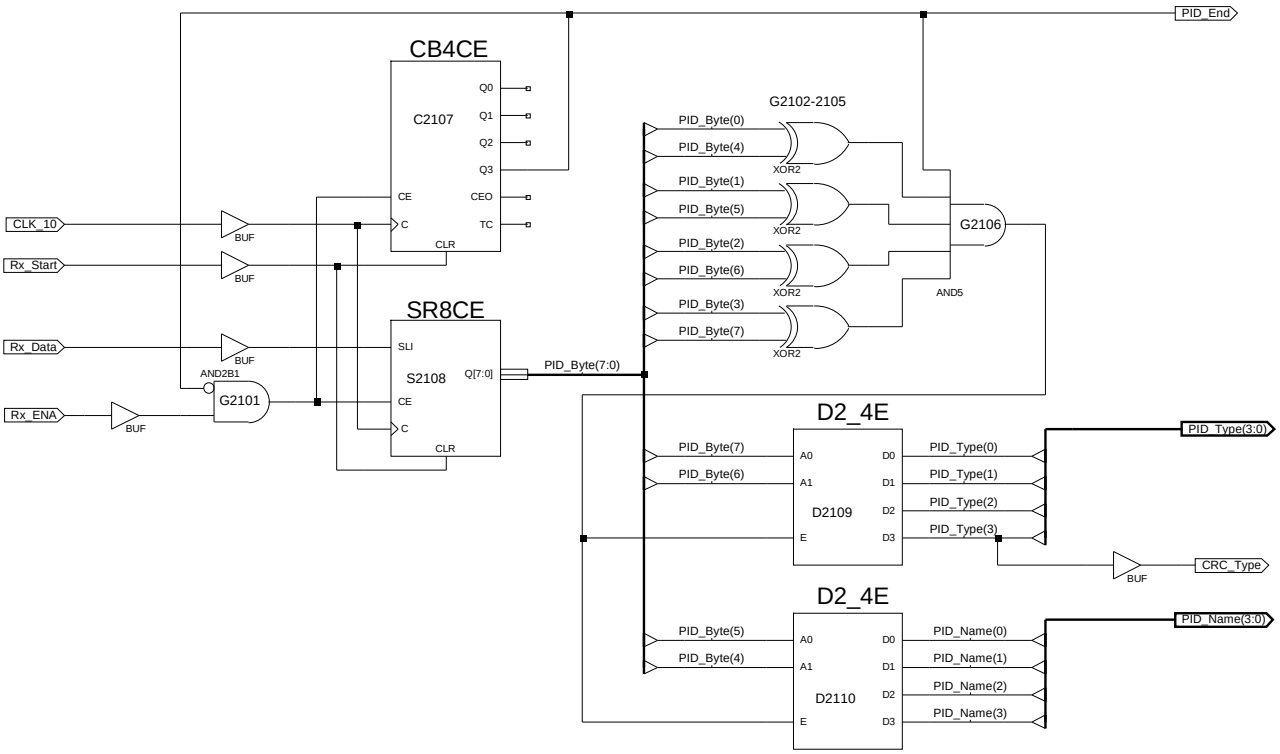


Rysunek 34: Schemat blokowy Stosu USB (**Stack**)

7 Opis działania bloków funkcjonalnych stosu USB

7.1 Blok kontroli poprawności i typu nagłówka pakietu (St_PID)

Na rysunku (35) przedstawiono schemat ideowy bloku zajmującego się identyfikacją typu pakietu na podstawie zajmującego pierwsze 8 bitów nagłówka (PID). Sygnał *Rx_Start*, generowany przez Transceiver w chwili zakończenia odbioru preambuły, zeruje licznik C2107 i rejestr przesuwny S2108 przygotowując je do odbioru nowego pakietu. Równocześnie z resetem licznika C2107 w stan niski przechodzi linia *PID_End* informująca o zakończeniu odbioru nagłówka. Sygnał wysoki na linii *Rx_ENA*, oznaczający zezwolenie ze strony Transceiver-a na przepisywanie odbieranych danych, poprzez bramkę G2101 i stan niski linii *PID_End*, zezwala licznikowi C2107 na zliczanie zboczy narastających sygnału zegarowego *CLK_10*, zaś rejestrowi S2108 na wpisywanie na nich kolejnych bitów pakietu z linii *Rx_Data*. Po odebraniu 8 bitów na wyjściu *Q3* licznika C2107 (linia *PID_End*) pojawia się stan wysoki blokując dalszą pracę licznika i rejestru przesuwnego. Równocześnie stan ten, poprzez bramkę G2106, umożliwia zadziałanie dekoderów D2109 i D2110, pod warunkiem pozytywnego wyniku kontroli poprawności odebranego nagłówka przeprowadzanej za pomocą bramek G2102-2105.



Rysunek 35: Schemat ideowy bloku kontroli poprawności i typu nagłówka pakietu (**St_PID**).

Ponieważ rodzaj obliczanego cyklicznego kodu nadmiarowego (CRC5 lub CRC16) determinuje typ pakietu zawarty w nagłówku, musi być on wyposażony w niezależny od kodu CRC mechanizm kontroli. Rzeczywistą informację o jednym z szesnastu typów pakietów zawierają, w zupełności do tego wystarczające, pierwsze (młodsze) cztery bity nagłówka, zaś pozostałe, starsze cztery bity stanowią ich negację (np. 'Token OUT' w notacji MSB→LSB: 1110-0001). Stąd też do kontroli poprawności wykorzystano bramki XOR porównujące ze sobą odpowiednie bity nagłówka (odpowiednio - zerowy z czwartym, pierwszy z piątym, itd.). Jeżeli odpowiadające sobie bity są różne, co może nastąpić tylko w takim przypadku, gdy jeden stanowi negację

drugiego, wynikiem wykonanej na nich operacji logicznej XOR będzie jedynka. W przypadku błędu transmisji, jeden z bitów będzie równy swojemu odpowiednikowi, co spowoduje wystawienie stanu niskiego na odpowiadającej im bramce XOR. Jeżeli bajt nagłówka został odebrany poprawnie, wyjścia wszystkich bramek G2102-2105 znajdą się w stanie wysokim, co w połączeniu z logiczną jedynką na linii *PID_End* powoduje wystawienie przez bramkę G2106 sygnału umożliwiającego pracę dekoderom D2109 i D2110.

W tabeli 1 przedstawiono tablicę prawdy dekodera typu "2 do 4". Zastosowanie dwóch dekoderów dwubitowych wynika z wprowadzonego przez specyfikację podziału rodzajów nagłówków na cztery główne typy (PID Type), każdy zawierający cztery podtypy (PID Name)[13]. Dzięki takiemu rozwiązaniu zmniejszona została ilość sygnałów wyjściowych układu służących identyfikacji rodzaju pakietu przez maszynę stanów **St_FSM**.

Wejścia			Wyjścia			
A1	A0	E	D3	D2	D1	D0
X	X	0	0	0	0	0
0	0	1	0	0	0	1
0	1	1	0	0	1	0
1	0	1	0	1	0	0
1	1	1	1	0	0	0

Tabela 1: Tabela prawdy dekodera "2 do 4".

Informację o typie nagłówka zawierają dwa najmłodsze bity (PID0 i PID1), podczas gdy podtyp zawarty jest w dwóch kolejnych bitach PID2 i PID3. Ponieważ każdy bajt przesyłany jest przez magistralę USB w kolejności od najmłodszego do najstarszego bitu, tak więc PID0 i PID1 odpowiadają liniom *PID_Byte(7)* i *PID_Byte(6)* magistrali *PID_Byte(7:0)* stanowiącej wyjście rejestru przesuwnego S2108. W zależności od kombinacji wartości bitów PID0 i PID1 dekoder D2109 ustawia w stan wysoki jedną z linii magistrali *PID_Type(3:0)*. Znaczenie poszczególnych kombinacji przedstawiono w tabeli 2. Czcionką pochyłą oznaczono typy nie wykorzystywane w trybie Full-Speed.

PID_Byte(6)	*PID_Byte(7)*	*PID_Type(i)*				Typ pakietu
PID1	PID0	3	2	1	0	(PID Type)
0	0	0	0	0	1	*Special*
0	1	0	0	1	0	Token
1	0	0	1	0	0	Handshake
1	1	1	0	0	0	Data

Tabela 2: Działanie dekodera D2109 (PID Type).

Wartości bitów PID2 i PID3 (sygnały *PID_Byte(5)* i *PID_Byte(4)*) są dekodowane przez dekoder D2110, w wyniku czego w stan wysoki zostaje ustawiona jedna z linii magistrali *PID_Name(7:0)*.

[13]Universal Serial Bus Specification, rev. 2.0, Tabela 8-1, s. 196

Analogicznie jw. znaczenie poszczególnych sygnałów przedstawiono w tabeli 3 z wyszczególnieniem nieużywanych podtypów. Pominięto nie używany w transmisji typ 'Special'. Ponieważ szesnastobitowy cykliczny kod nadmiarowy CRC16 zawierają jedynie pakiety typu 'Data' (pozostałe zawierają pięciobitowe CRC5, bądź, jak 'Handshake', nie posiadają kodu CRC), do identyfikacji typu obliczanego w dalszej kolejności kodu użyto linii *PID_Type(3)* wyprowadzonej z bloku pod nazwą *CRC_Type*. Stan wysoki tej linii, oznaczający odbiór pakietu typu 'Data', informuje blok **St_RxCRC** o konieczności obliczania kodu szesnastobitowego, stan niski zaś - pięciobitowego.

Typ pakietu						Token	Handshake	Data
PID_Byte(4)	*PID_Byte(5)*	*PID_Name(i)*				Podtyp pakietu		
PID3	PID2	3	2	1	0	(PID Name)		
0	0	0	0	0	1	OUT	ACK	DATA0
0	1	0	0	1	0	*SOF*	*NYET*	*DATA2*
1	0	0	1	0	0	IN	NAK	DATA1
1	1	1	0	0	0	SETUP	*STALL*	*MDATA*

Tabela 3: Działanie dekodera D2110 (PID Name).

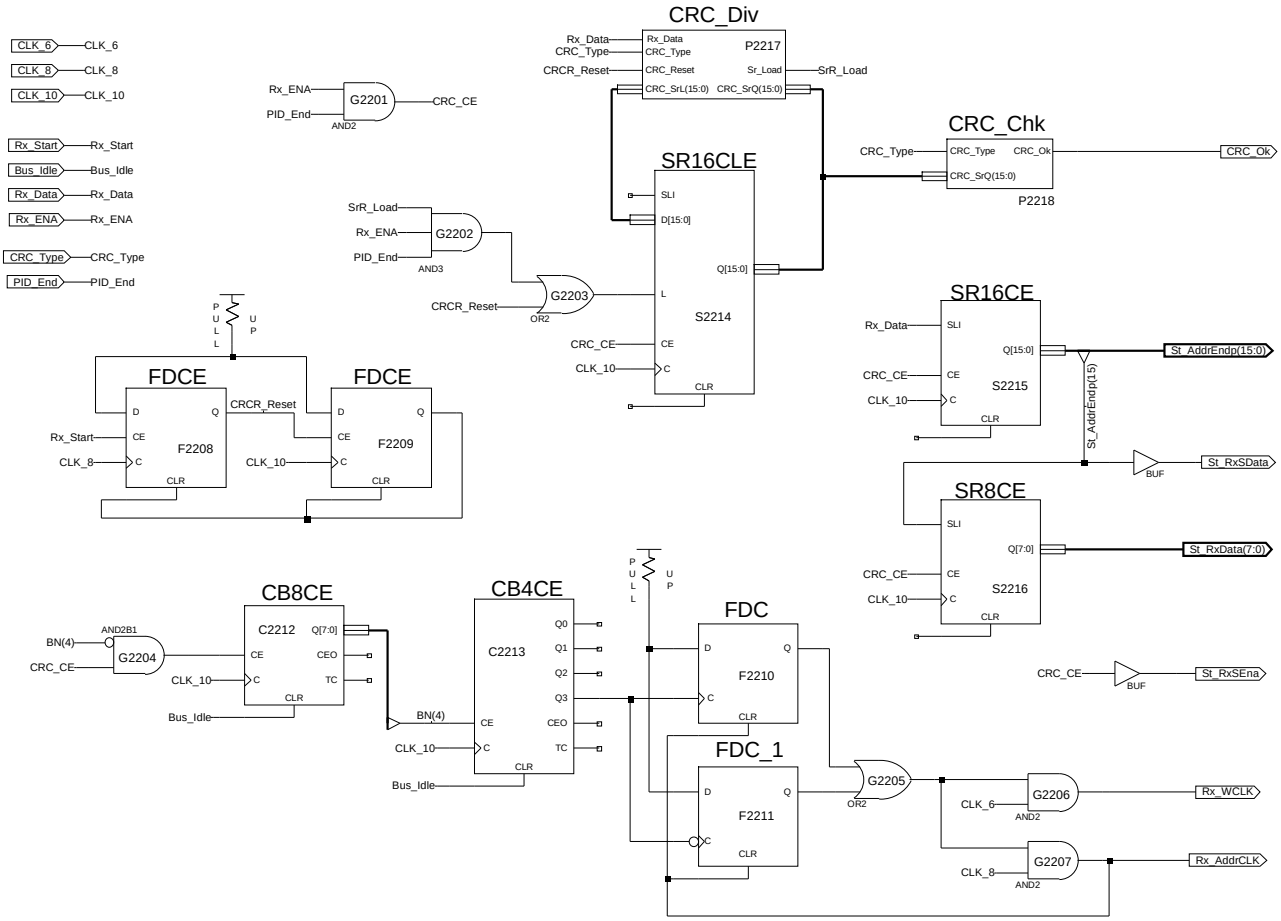
7.2 Blok obliczania sumy kontrolnej CRC pakietów odbieranych (St_RxCRC)

Rysunek (36) przedstawia blok obliczający i kontrolujący poprawność cyklicznego kodu nadmiarowego CRC. Dodatkowym zadaniem bloku jest usunięcie znajdującego się na końcu danych kodu oraz konwersja danych przychodzących w postaci transmisji szeregowej na ośmiobitową transmisję równoległą.

Elementem przechowującym aktualny wynik obliczeń kodu jest szesnastobitowy rejestr przesuwny S2214 z możliwością równoległego zapisu wszystkich komórek rejestru jednocześnie. Jeżeli na wejściu *CE* rejestru występuje stan wysoki a na *L* niski, rejestr działa w sposób klasyczny, przesuwając zawartość na zboczu narastającym sygnału zegarowego *C* i wpisując w miejsce najmłodszego bitu stan wejścia *SLI*. W przypadku gdy wejście *L* znajduje się w stanie wysokim (niezależnie od stanu wejścia *CE*), na zboczu zegara cała zawartość rejestru zostaje zastąpiona przez wartości podane na wejście *D[15:0]*.

Zgodnie z algorytmem obliczania kodu zawartym w specyfikacji[14], pierwszą czynnością jest ustawienie wszystkich bitów rejestru S2214 w stan wysoki. Logiczna jedynka na linii *Rx_Start* umożliwia przerzutnikowi F2208 ustawienie linii *CRCR_Reset* w stan wysoki na pierwszym zboczu zegara *CLK_8*. W wyniku tego blok **CRC_Div** ustawia wszystkie linie wyjściowej magistrali *CRC_SrL(15:0)* w stan wysoki (blok ten stanowi wyjątek - ze względu na czytelność połączeń wejściową magistralę *CRC_SrQ(15:0)* umieszczono po prawej, a wyjściową po lewej stronie symbolu). Stan wysoki na linii *CRCR_Reset* poprzez bramkę G2203 ustawia logiczną jedynkę na wejściu *L* rejestru, co oznacza zezwolenie na wpisanie do rejestru wartości ustawionych na magistrali *CRC_SrL(15:0)*. Następuje to na zboczu sygnału zegarowego *CLK_10*, które

[14]Universal Serial Bus Specification, rev. 2.0, s. 198

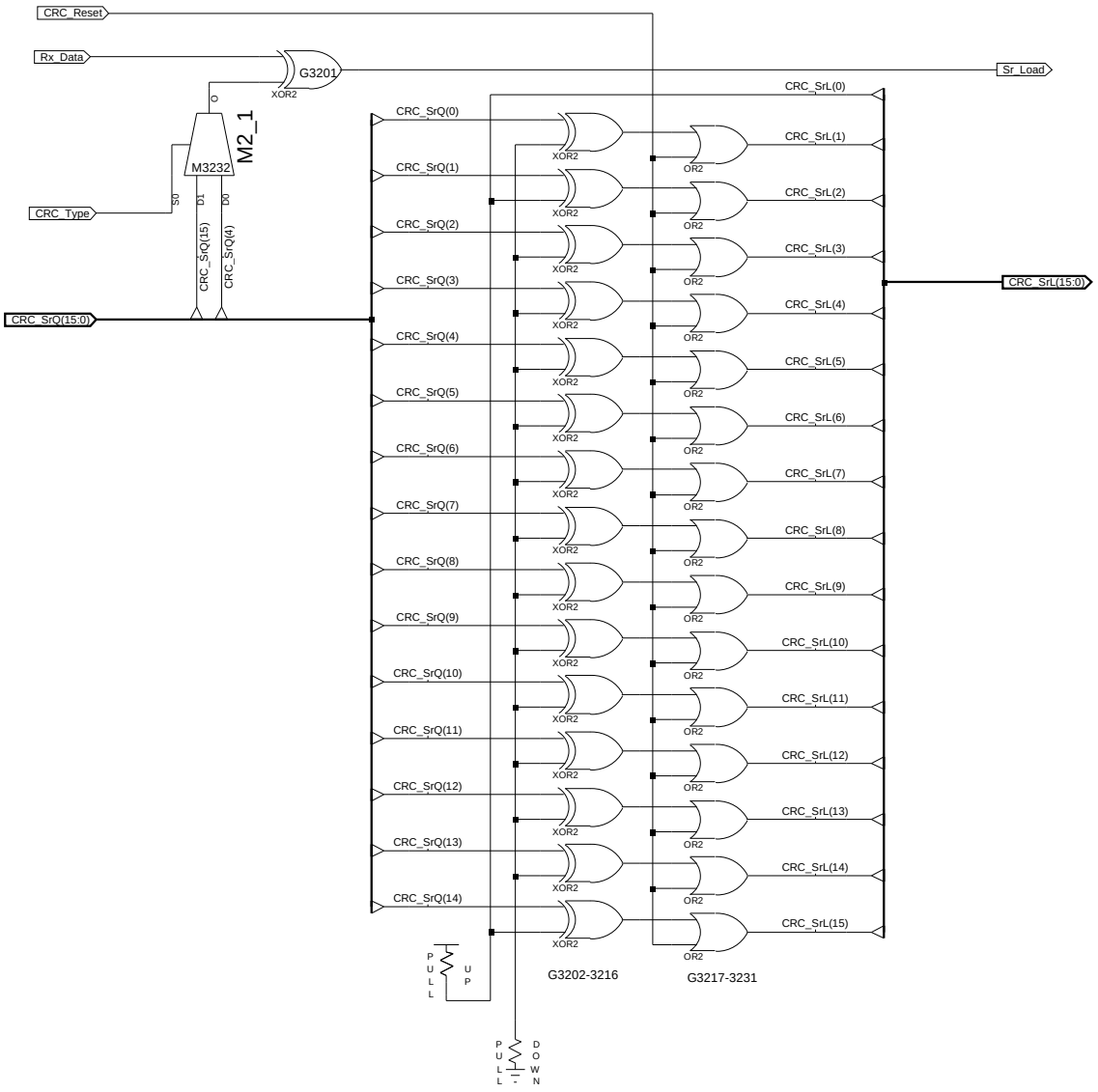


Rysunek 36: Schemat ideowy bloku obliczania CRC pakietów odebranych (**St_RxCRC**).

równocześnie powoduje ustawienie wyjścia *Q* przerzutnika F2209 w stan wysoki, a, co za tym idzie, zresetowanie obu przerzutników i powrót linii *CRCR_Reset* do stanu niskiego.

Po zakończeniu odbioru nagłówka przez blok **St_PID** sygnał *PID_End* przechodzi w stan wysoki. Występująca równocześnie na linii *Rx_ENA* logiczna jedynka (zezwolenie na przepisywanie danych z Transceiver-a) ustawia przez bramkę G2201 stan wysoki na linii *CRC_CE* umożliwiając pracę rejestru S2214 oraz rejestrów S2215 i S2216 omówionych dalej. Równocześnie występujące stany wysokie na liniach *PID_End* i *Rx_ENA* umożliwiają także ustawienie logicznej jedynki na wejściu *L* rejestru poprzez bramki G2202 i G2203, pod warunkiem wystąpienia stanu wysokiego na linii *SrR_Load*. Tak więc wraz z wystawieniem przez Transceiver nowego bitu danych (*Rx_Data*) rejestr S2214 jest przesuwany z wpisaniem zera na najmłodszy bit bądź ładowany wartością magistrali *CRC_SrL(15:0)* w zależności od stanu linii *SrR_Load*. Sygnał ten jest tworzony w bloku **CRC_Div** na podstawie porównania bitu danych z linii *Rx_Data* z wartością najstarszego bitu sumy kontrolnej pobieranej z rejestru S2214 poprzez magistralę *CRC_SrQ(15:0)*. Na rysunku (37) przedstawiono schemat ideowy dzielnika sumy **CRC_Div**. Poprzez bramkę G2301 wykonywana jest operacja logiczna XOR na bicie danych (*Rx_Data*) oraz najstarszym bicie sumy (wyjście *O* multipleksera M3232). Multiplekser służy do wyboru pomiędzy pięcio- i szesnastobitowym kodem CRC podając na wyjście *O* piąty (*CRC_SrQ(4)*, stan niski *CRC_Type*) bądź szesnasty bit (*CRC_SrQ(15)*, *CRC_Type* w stanie wysokim) sumy podawanej z rejestru S2214 poprzez magistralę *CRC_SrQ(15:0)*. Jeżeli wynikiem operacji XOR jest jedynka, rejestr S2214 jest ładowany wartością tworzoną za pomocą bramek G3202-3216. Zgodnie ze specyfikacją, jeśli bit danych jest różny od najstarszego bitu sumy, wartość rejestru

powinna zostać przesunięta o jeden bit w przód, a następnie "podzielona" przez wielomian generacyjny kodu poprzez wykonanie operacji XOR na każdym z bitów sumy i odpowiadającym mu bicie wielomianu (odpowiednio - pierwszy bit sumy XOR pierwszy bit wielomianu, itd.). Jako że wielomian generacyjny kodu pięciobitowego CRC5 (postaci 00101B) jest równy początkowym bitom wielomianu kodu szesnastobitowego CRC16 (1000000000000101B), budowa dwóch osobnych bloków dla obu kodów nie była konieczna.

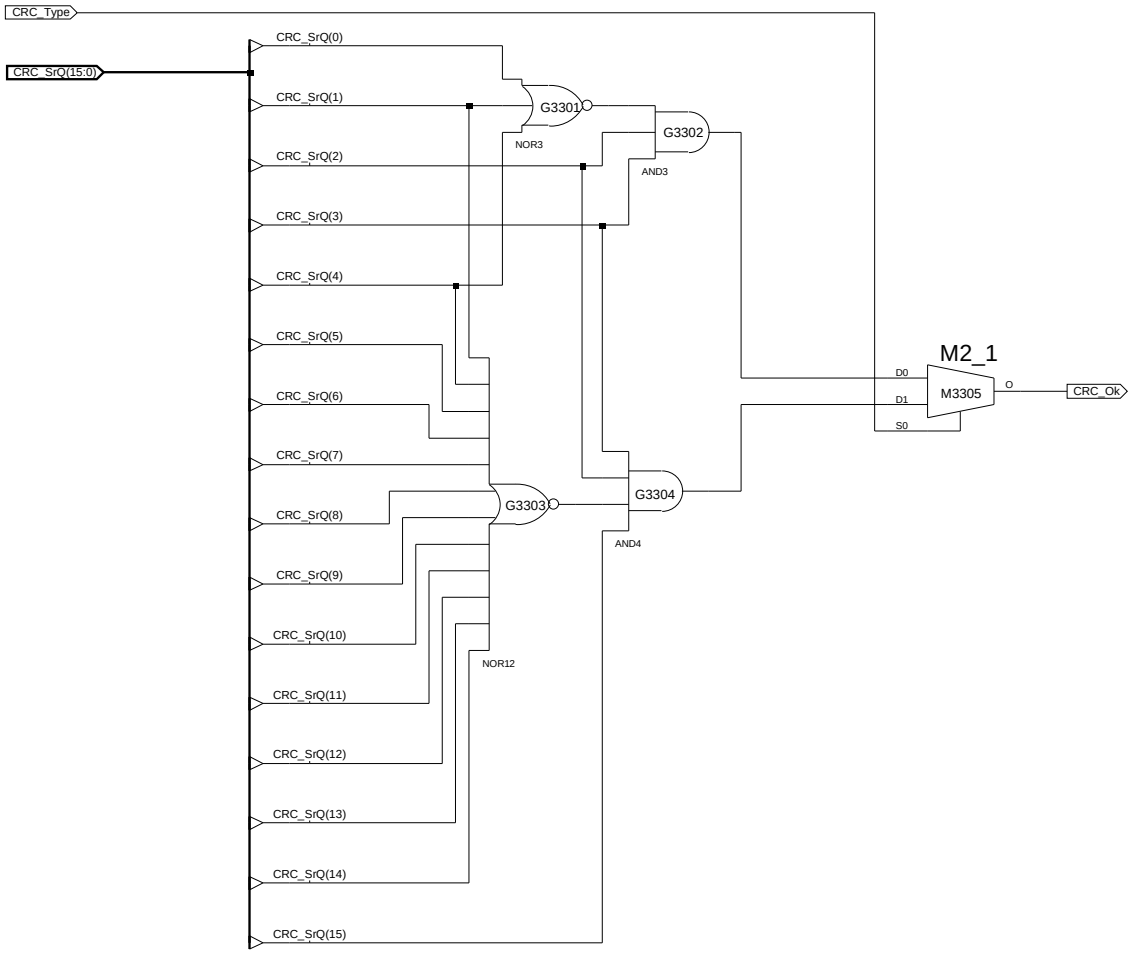


Rysunek 37: Dzielenie przez wielomian generacyjny CRC - schemat ideowy (**CRC_Div**).

Ponieważ nie ma możliwości jednoczesnego przesunięcia i załadowania wartości do rejestru S2214 na jednym zboczu zegara *CLK_10*, aby uniknąć tworzenia dodatkowego układu generacji sygnału zegarowego o dwóch zboczach narastających w obrębie jednego cyklu, przesunięcie rejestru wykonano poprzez odpowiednie połączenie linii magistrali wyjściowej i wejściowej. Najmłodszy (pierwszy) bit sumy (*CRC_SrQ(0)*) pobierany z wyjścia rejestru S2214, poddawany jest operacji XOR z drugim bitem wielomianu generacyjnego i wpisywany do drugiej komórki rejestru S2214 poprzez linię *CRC_SrL(1)*. Analogicznie przesuwane są pozostałe bity z pominięciem ostatniego, szesnastego bitu (*CRC_SrQ(15)*), usuwanego z rejestru podczas operacji przesunięcia. Najmłodszy bit (*CRC_SrL(0)*) wpisywany do rejestru S2214 jest zawsze równy

1, ponieważ w operacji przesunięcia przyjmuje on zawsze wartość 0 poddawaną alternatywie wykluczającej z 1 najmłodszego bitu obu wielomianów. Bramki G3217-3231 odpowiadają za ustawienie wszystkich linii magistrali *CRC_SrL(15:0)* w stan wysoki podczas wstępnego ładowania rejestru opisanego wyżej (linia *CRC_Reset* znajduje się wówczas w stanie wysokim).

Po poprawnym odebraniu całego pakietu (wszystkich danych oraz odpowiedniej ilości bitów cyklicznego kodu nadmiarowego na końcu) w rejestrze powinna pozostać wartość nazywana resztą sumy kontrolnej, równa 01100B dla CRC5 lub 1000000000001101B dla CRC16. Wystąpienie innej wartości na magistrali *CRC_SrQ(15:0)* oznacza wystąpienie błędu podczas transmisji pakietu. Sprawdzaniem poprawności reszty sumy kontrolnej zajmuje się blok **CRC_Chk**, którego schemat ideowy przedstawiono na rysunku (38).



Rysunek 38: Kontrola poprawności reszty sumy kontrolnej - schemat ideowy (**CRC_Chk**).

Bramki G3301 (CRC5) i G3303 (CRC16) wykonują operację zanegowanej sumy logicznej na tych bitach reszty, na których powinno występować logiczne zero. Jeśli tak jest, to wyjście danej bramki znajduje się w stanie wysokim, co w połączeniu z pozostałymi bitami reszty sumy, równymi jeden przy poprawnej transmisji, powoduje ustawienie logicznej jedynki na wyjściu bramek G3302 (CRC5) i G3304 (CRC16). Stan wysoki na wyjściu danej bramki oznacza poprawną transmisję pakietu o danej długości kodu nadmiarowego. Informacja ta jest przekazywana na wyjście układu *CRC_Ok* poprzez multiplekser M3305 wybierający, w zależności od stanu linii *CRC_Type*, wyjście bramki G3302 lub G3304. Blok **CRC_Div** działa w sposób ciągły (także podczas transmisji pakietu i obliczania sumy), jednakże stan linii *CRC_Ok* jest sprawdzany przez maszynę stanów **St_FSM** dopiero po zakończeniu odbioru.

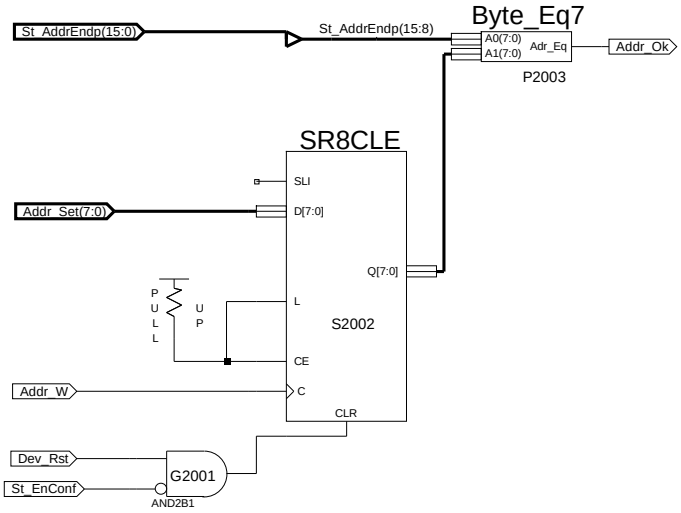
Równolegle z pracą rejestru S2214 obliczającego sumę kontrolną, dane są przesyłane do rejestru S2215, usuwającego z pakietu kod CRC. Ponieważ pakiet danych (typu 'Data') posiada zmienny rozmiar, nie ma możliwości odcięcia kodu CRC poprzez zablokowanie transmisji danych do kolejnych bloków po odliczeniu określonej liczby bajtów. Z tego względu zdecydowano się na zastosowanie pośredniczącego w transmisji rejestru S2215 o rozmiarze sumy kontrolnej CRC16. Dane konwertowane dalej na ośmiobitową transmisję równoległą w rejestrze S2216 są więc opóźnione o szesnaście cykli zegara względem rzeczywistego momentu ich odbioru, lecz, dzięki takiemu rozwiązaniu, dwa bajty kodu CRC16 zostają zatrzymane w rejestrze w chwili zakończenia transmisji i nie są przesyłane do dalszych bloków. Dodatkowo, w przypadku odbioru pakietu typu 'Token', siedmiobitowy adres interfejsu, czterobitowy numer bufora ('Endpoint') oraz kończące transmisję CRC5 (łącznie szesnaście bitów) zostają zgromadzone w rejestrze S2215 i przesłane przez magistralę *St_AddrEndp(15:0)* do bloku kontrolującego zgodność adresu pakietu i interfejsu **St_Addr**. Najstarszy bit rejestru S2215 *St_AddrEndp(15)*, czyli dane zawarte w pakiecie typu 'Data', są przesyłane w postaci szeregowej przez linię *St_RxSData* wraz z sygnałem zezwolenia *St_RxSEna* do bloku interpretacji rozkazów konfiguracyjnych **St_Brqst** oraz konwertowane do postaci równoległej (magistrala *St_RxData(7:0)*) poprzez rejestr S2216. Liczniki C2212 i C2213 wraz z przerzutnikami F2210, F2211 odpowiadają za generację sygnałów zegarowych do zapisu danych z magistrali równoległej. Oba liczniki C2212 i C2213 są resetowane stanem wysokim linii *Bus_Idle* (brak jakiejkolwiek transmisji na magistrali USB). W chwili rozpoczęcia odbioru danych piąty bit licznika C2212 (sygnał *BN(4)*) znajduje się więc w stanie niskim, co w połączeniu ze stanem wysokim sygnału *CRC_CE* umożliwia przez bramkę G2204 zliczanie odbieranych bitów. Po odebraniu dwóch bajtów (rejestr S2215 zapełniony, pierwszy bit danych wystawiony na linii *St_AddrEndp(15:0)*) linia *BN(4)* przechodzi w stan wysoki blokując licznik C2212 i, jednocześnie, odblokowując licznik C2213. Co każde odebrane osiem bitów (co odpowiada utworzeniu jednego pełnego bajtu przez rejestr S2216) wyjście *Q3* licznika C2213 zmienia stan na przeciwny. Przerzutniki F2210 i F2211 wykrywają zbocza (odpowiednio - narastające i opadające) na tym sygnale, wystawiając na swoich wyjściach *Q* stan wysoki. Sygnały z obu przerzutników są sumowane przez bramkę G2205 i podawane na wejścia bramek G2206 i G2207 kluczujących przebiegi zegarowe. Bajt danych, a więc i stan wysoki na wyjściu bramki G2205, tworzony jest na zboczu zegara *CLK_10*. Pojawiające się jako pierwsze w następnym cyklu zegara zbocze narastające na sygnale *CLK_6*, poprzez bramkę kluczującą G2206 przekazywane jest linią *Rx_WCLK* do bufora powodując zapis danych z magistrali *St_RxData(7:0)*. Nadchodzący jako następny i przekazywany przez bramkę G2207, sygnał zegarowy *CLK_8* zwiększa licznik adresowy pamięci w buforze (linia *Rx_AddrCLK*) oraz powoduje skasowanie przerzutników F2210 i F2211 blokując bramki kluczujące aż do sformowania kolejnego bajtu danych w rejestrze S2216.

7.3 Blok sprawdzania zgodności adresu interfejsu (St_Addr)

Na rysunku (39) przedstawiono schemat ideowy bloku sprawdzającego zgodność adresu otrzymanego w pakiecie z aktualnym adresem interfejsu. Aktualny adres interfejsu przechowywany jest w rejestrze S2002. Zgodnie z procesem enumeracji[15] interfejs, do czasu pomyślnego zakończenia etapu konfiguracji (sygnalizowanego stanem wysokim sygnału *St_EnConf* z bloku **St_Enum**), powinien odpowiadać ustawieniem podstawowego adresu 00H na każdy sygnał resetu komunikowany stanem wysokim linii *Dev_Rst*. Jest to zapewnione poprzez czyszczenie zawartości rejestru S2002 poprzez bramkę G2001. Otrzymany w procesie konfiguracji adres jest podawany z bloku interpretacji rozkazów **St_Brqst** magistralą *Addr_Set(7:0)* i zapisywa-

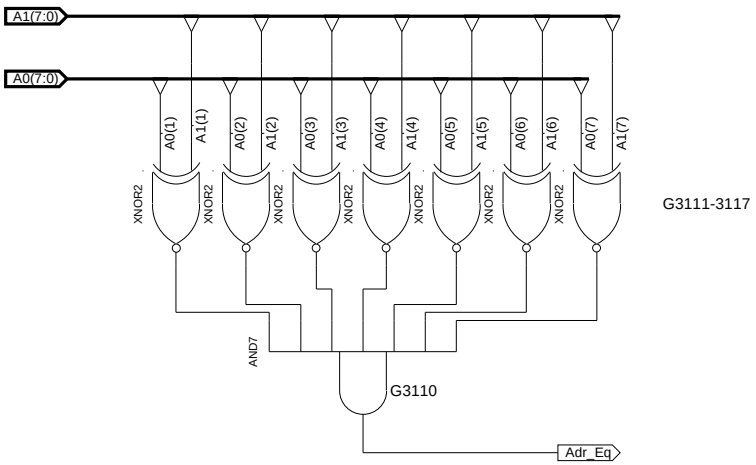
[15] Universal Serial Bus Specification, rev. 2.0, s. 241

ny w rejestrze S2002 zboczem narastającym sygnału *Addr_W* generowanym również w bloku **St_Brqst**.



Rysunek 39: Schemat ideowy bloku sprawdzania zgodności adresu interfejsu (**St_Addr**).

Porównaniem wartości adresów przechowywanego w rejestrze S2002 i odebranego w pakiecie typu 'Token' zajmuje się blok **Byte_Eq7**, którego schemat ideowy przedstawiono na rysunku (40). Adres zawarty w pakiecie zajmuje siedem jego pierwszych bitów, stąd z magistrali *St_AddrEndp(15:0)* do porównywania pobierane jest osiem najstarszych bitów *St_AddrEndp(15:8)* (ze względu na konstrukcję magistral w *ISE* korzystniej było pobrać magistralę o wymiarze takim, jak wyjście rejestru przesuwnego i pominąć podczas sprawdzania zbędny najmłodszy bit).

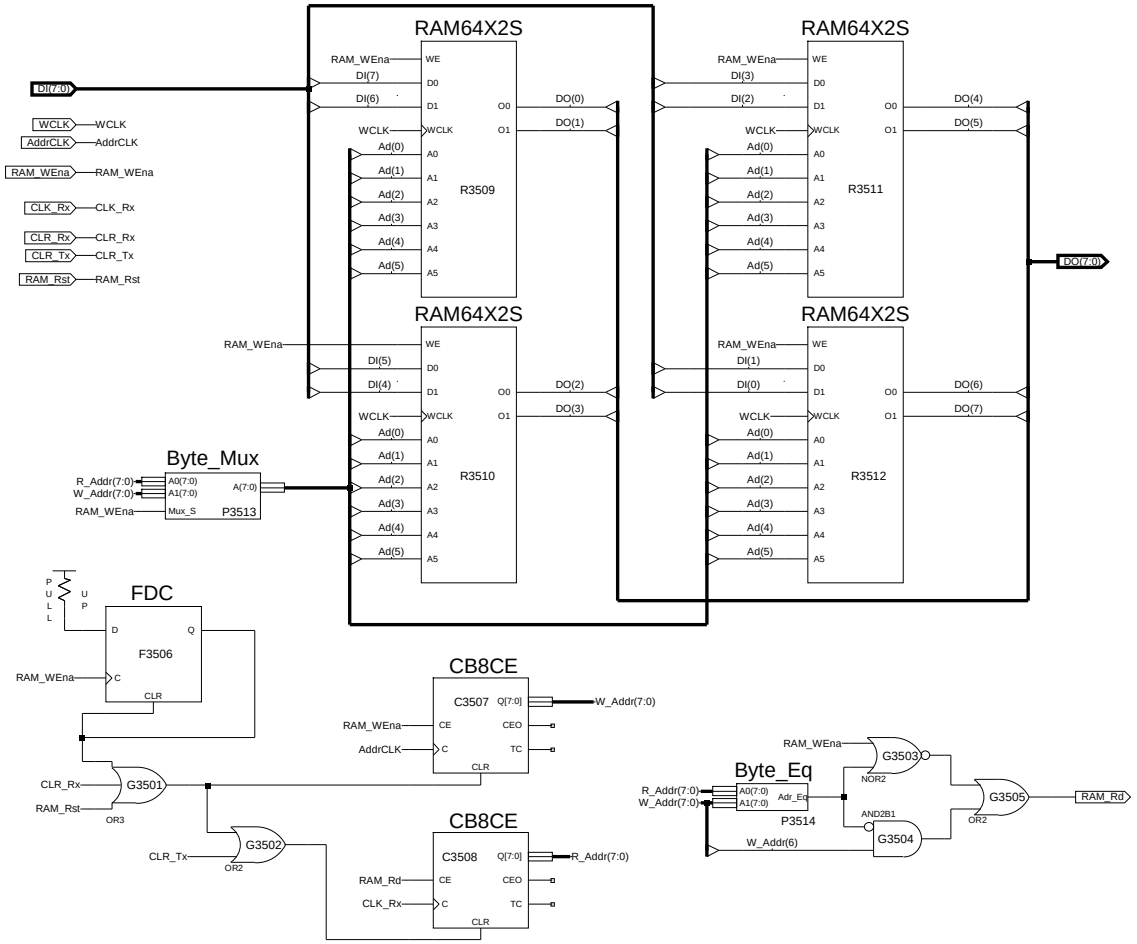


Rysunek 40: Porównywanie wartości dwóch magistral siedmiobitowych (**Byte_Eq7**).

Poszczególne bity (z pominięciem najmłodszych) obu magistral *A0(7:0)* i *A1(7:0)* są porównywane za pomocą bramek G3111-3117 realizujących zanegowaną alternatywę wykluczającą XNOR. Jeśli bity we wszystkich parach są sobie równe, wyjścia wszystkich bramek G3111-3117 są w stanie wysokim, a, co za tym idzie, w stanie wysokim znajduje się linia *Adr_Eq* będąca wyjściem bramki G3110. Podobnie jak w przypadku bloku **CRC_Chk** sprawdzanie adresu odbywa się w sposób ciągły, jednak sygnał *Addr_Ok* wystawiany w stan wysoki przy równości obu adresów jest sprawdzany dopiero po zakończeniu transmisji.

7.4 Pamięć RAM 64B (St_RAM64)

Rysunek (41) przedstawia schemat ideowy bloku pamięci. Ze względu na zamierzoną uniwersalność projektu nie zastosowano dedykowanego dla układu *Spartan3A* gotowego bloku pamięci, projektując w jego miejsce blok zawierający elementy wspólne dla całej rodziny układów Xilinx.

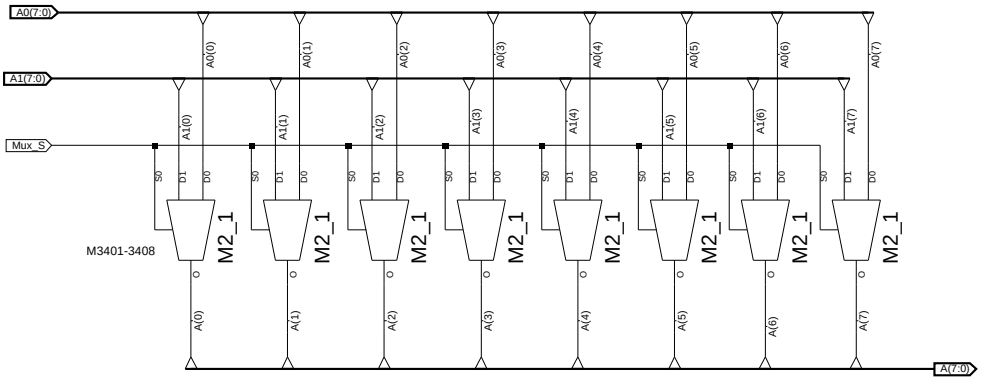


Rysunek 41: Pamięć RAM 64B - schemat ideowy (**St_RAM64**).

Pamięć o rozmiarze 64 słów ośmiobitowych zbudowano w oparciu o cztery bloki pamięci R3509-3512 mieszczącej 64 słowa dwubitowe. Każdy z bloków adresowany jest za pomocą sześciu młodszych bitów ośmiobitowej magistrali adresowej *Ad(7:0)*.Zapis do zaadresowanej w ten sposób komórki pamięci danych dostarczonych do wejść *D0* i *D1* następuje na zboczu narastającym zegara *WCLK* pod warunkiem wystąpienia stanu wysokiego na wejściu *WE* bloku. Odczyt zawartości pamięci z wyjść *O0* i *O1* następuje asynchronicznie po podaniu odpowiedniego adresu. Ze względu na stosowany w standardzie USB format transmisji bajtu od LSB do MSB, magistrale doprowadzająca dane przeznaczone do zapisu (*DI(7:0)*) oraz magistrala stanowiąca wyjście pamięci (*DO(7:0)*) zostały "skrzyżowane", tzn. najstarszy bit magistrali wejściowej (*DI(7)*) odpowiada najmłodszemu wyjściowej (*DO(0)*), itd. Dzięki takiemu rozwiązaniu komunikacja z urządzeniem zewnętrznym następuje w bardziej intuicyjnym standardzie, gdy najstarszy bit magistrali odpowiada MSB bajtu danych.

Za adresowanie pamięci podczas zapisu odpowiada licznik C3507. Podczas gdy zewnętrzny sygnał zezwalający na zapis pamięci (*RAM_WEna*) znajduje się w stanie wysokim, licznik C3507 zwiększa adres wystawiany na magistralę *W_Addr(7:0)* na każdym zboczu narastającym

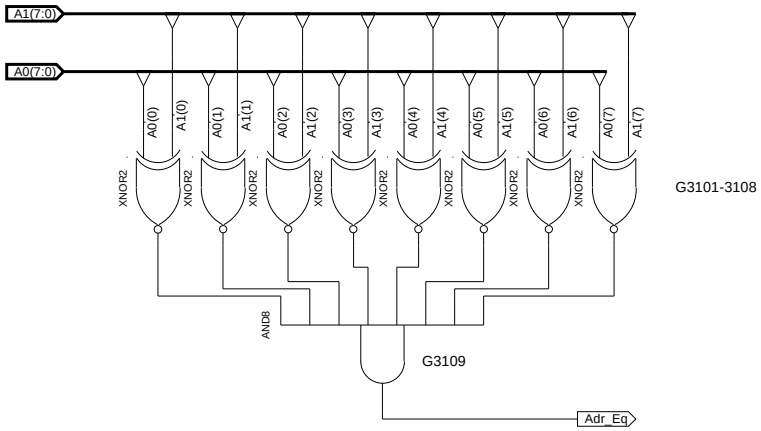
sygnału zegarowego *AddrCLK*. Po zakończeniu procesu zapisu licznik przechowuje adres o jeden wyższy od ostatniej zapisanej komórki umożliwiając identyfikację rozmiaru zapisanych w pamięci danych. Magistrala *W_Addr(7:0)* jest podczas procesu zapisu podpięta do magistrali adresowej pamięci *Ad(7:0)* poprzez multiplekser jednobajtowy **Byte_Mux**, którego schemat ideowy przedstawiono na rysunku (42).



Rysunek 42: Multiplekser jednobajtowy - schemat ideowy (**Byte_Mux**).

Zestaw ośmiu multiplekserów M3401-3408 dołącza do kolejnych bitów magistrali *A(7:0)* odpowiednie bity magistral *A0(7:0)* lub *A1(7:0)*, w zależności od stanu sygnału *Mux_S*. W stanie wysokim tego sygnału do magistrali wyjściowej dołączona jest magistrala *A1(7:0)*, co odpowiada podpięciu licznika C3507 do magistrali adresowej pamięci. Po zakończeniu zapisu linia *RAM_WEna* przechodzi w stan niski umożliwiając adresowanie pamięci licznikowi C3508 odpowiedzialnemu za odczyt danych. Licznik ten, uzyskując zezwolenie stanem wysokim na linii *RAM_Rd*, oznaczającym gotowość pamięci do odczytu, zlicza kolejne takty sygnału zegarowego *CLK_Rx* zwiększając adres odczytywanej komórki pamięci (magistrala *R_Addr(7:0)*). Oba liczniki są ustawiane w stan początkowy 00H poprzez przerzutnik F3506 generujący krótki impuls kasujący wskazania liczników na zboczu narastającym sygnału *RAM_WEna*, czyli na samym początku zapisu. Sygnał kasowania jest łączony przez bramkę G3501 z dwoma zewnętrznymi sygnałami pełniącymi taką samą funkcję. Sygnał *RAM_Rst* odpowiada za skasowanie liczników podczas resetu interfejsu, zaś sygnał *CLR_Rx* umożliwia usunięcie informacji o zapisaniu pamięci (np. po odebraniu pakietu danych o błędnym CRC). Licznik adresu odczytu C3508 posiada dodatkowo możliwość skasowania sygnałem *CLR_Tx*, co umożliwia ponowny odczyt zawartości pamięci przy wystąpieniu błędu transmisji nadawanego pakietu.

Proces odczytu pamięci trwa do momentu zrównania się wskazań obu liczników C3507 i C3508, co jest równoznaczne z odczytaniem zawartości wszystkich zapisanych komórek. Do porównywania wartości obu magistral adresowych służy blok **Byte_Eq** działający analogicznie jak przedstawiony wyżej **Byte_Eq7**. Schemat ideowy bloku **Byte_Eq** przedstawiono na rysunku (43). Wyjście bloku **Byte_Eq** przechodzi w stan niski z chwilą zapisu pierwszego bajtu do pamięci. Po zakończeniu zapisu sygnał *RAM_WEna* również przechodzi w stan niski umożliwiając przez bramki G3503 i G3505 wystawienie stanu wysokiego na linii *RAM_Rd* o znaczącego gotowość pamięci do odczytu. Stan ten utrzymuje się do momentu zrównania się wskazań liczników i, w konsekwencji, przejścia sygnału wyjściowego bloku **Byte_Eq** w stan wysoki a *RAM_Rd* w niski. Dodatkowym zabezpieczeniem przed przepełnieniem pamięci jest sygnał pochodzący z bramki G3504, który wymusza przejście linii *RAM_Rd* w stan wysoki z chwilą osiągnięcia przez licznik C3507 adresu 65 (stan wysoki na siódmym bicie *W_Addr(6)*). Stan ten utrzymuje się do chwili osiągnięcia przez licznik C3508 adresu 65 i pojawienia się sygnału wysokiego na wyjściu bloku **Byte_Eq**.

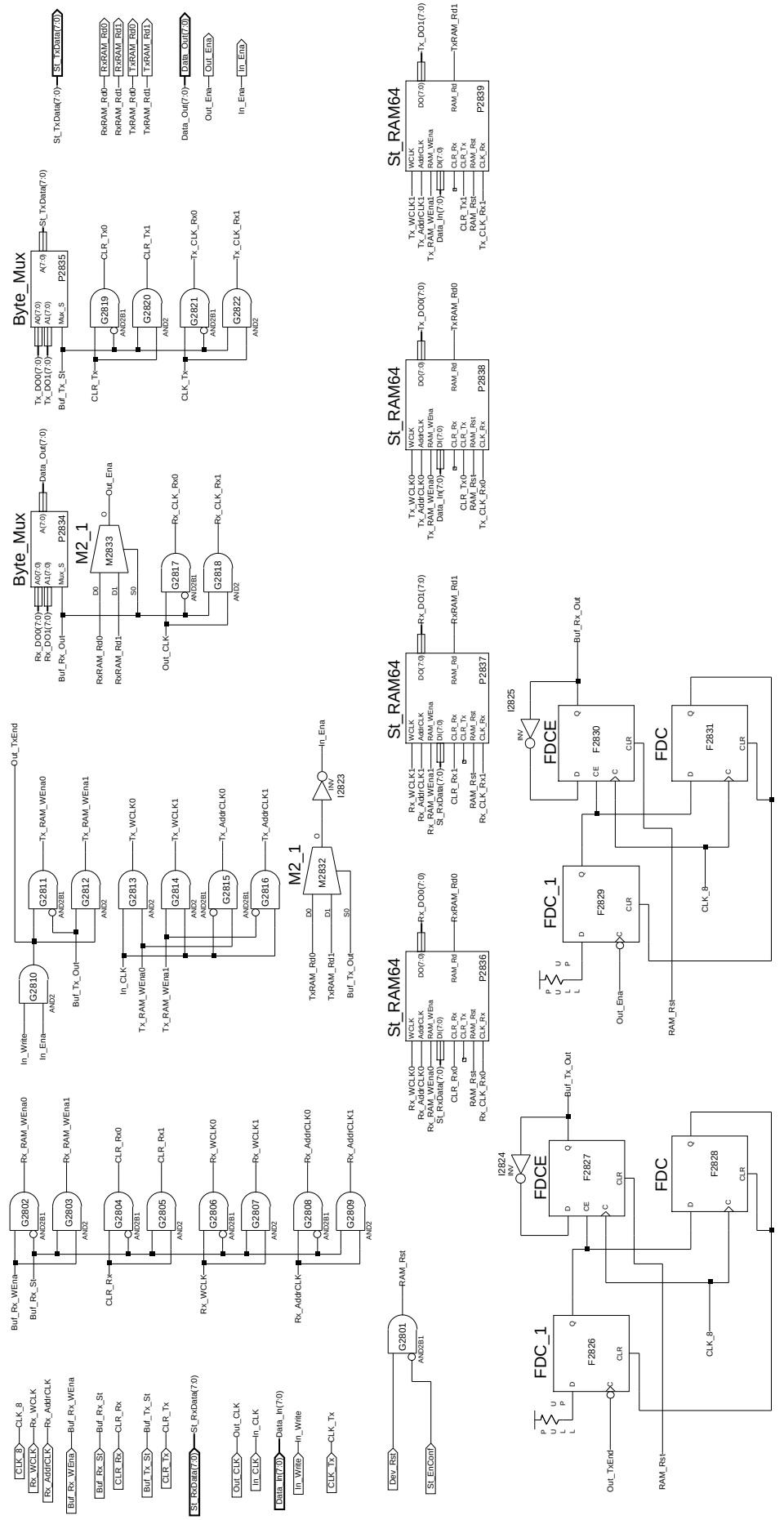


Rysunek 43: Porównywanie wartości dwóch bajtów (**Byte_Eq**).

7.5 Bufor wejścia - wyjścia danych (St_Buff)

Na rysunku (44) przedstawiono schemat ideowy bloku zapewniającego komunikację z zewnętrznym urządzeniem podpiętym do interfejsu, czyli bufora danych wejściowych i wyjściowych. Zawiera on cztery opisane wyżej bloki pamięci (**St_RAM64**) oraz logikę sterującą zapisem i odczytem danych, zarówno przez urządzenie, jak i interfejs. Zastosowano po dwa bloki pamięci na każdy z buforów w celu umożliwienia równoczesnego dostępu do bufora zarówno ze strony zewnętrznego urządzenia jak i interfejsu (przykładowo - urządzenie czyta uprzednio odebrane dane z pierwszego bloku pamięci podczas gdy stos ma możliwość zapisu aktualnie odbieranego pakietu danych do drugiego bloku). Dzięki takiemu rozwiązaniu, o ile tylko zewnętrzne urządzenie może odczytywać dane wystarczająco szybko, zawsze jeden z bloków pamięci bufora jest wolny i stos może odbierać każdy nadchodzący pakiet danych bez konieczności retransmisji z powodu zajętości bufora. Takie rozwiązanie skomplikowało jednak w znaczący sposób logikę sterującą buforami (zlokalizowaną po części w opisywanym bloku i maszynie stanów **St_FSM**). Komplikacja ta wyniknęła z konieczności przełączania bloków pamięci między stosem a urządzeniem w sposób z jednej strony niezależny (aby np. urządzenie mogło zapisać oba bloki bufora wejściowego, jeśli nie są używane), a z drugiej strony tak, aby nie wystąpił konflikt w postaci próby jednoczesnego zapisu i odczytu tego samego bloku lub zamiana kolejności pakietów danych.

O dostępie ze strony interfejsu do danego bloku pamięci decyduje maszyna stanów **St_FSM** poprzez wystawienie stanu niskiego lub wysokiego na liniach *Buf_Rx_St* dla bufora wyjściowego i *Buf_Tx_St* dla wejściowego. Magistrala danych *St_RxData(7:0)* przeznaczonych do zapisu w buforze wyjściowym jest dołączona bezpośrednio do obu bloków pamięci P2836 i P2837. Sterowanie zapisem w wybranym bloku odbywa się poprzez bramki G2802 i G2803 przepuszczające sygnał zezwalający na zapis *Buf_Rx_WEna* do wybranego stanem linii *Buf_Rx_St* bloku (*Rx_RAM_WEna0* dla pierwszego i *Rx_RAM_WEna1* dla drugiego). W analogiczny sposób poprzez bramki G2806 i G2807 odbywa się kluczowanie przebiegu zegarowego zapisującego dane *Rx_WCLK*, zwiększającego adres *Rx_AddrCLK* (bramki G2808 i G2809) oraz sygnału wyczyszczenia pamięci po niepoprawnym odbiorze pakietu *CLR_Rx* (G2804 i G2805). Maszyna stanów jest informowana o gotowości do odczytu zapisanego przez urządzenie zewnętrzne bloku bufora wejściowego przez linie *TxRAM_Rd0* i *TxRAM_Rd1*. Magistrale danych wyjściowych obu bloków P2838 i P2839 (odpowiednio, *Tx_DO0(7:0)* i *Tx_DO1(7:0)*) są dołączane do magistrali wyjściowej bufora *St_TxData(7:0)* przez blok **Byte_Mux** w zależności od stanu linii *Buf_Tx_St*.



Rysunek 44: Bufor wejścia - wyjścia - schemat ideowy (**St_Buff**).

Sygnał zegarowy zwiększający odczytywany adres *CLK_Tx* jest kluczowany przez bramki G2821 i G2822 i przesyłany do wybranego bloku pamięci liniami *Tx_CLK_Rx0* lub *Tx_CLK_Rx1*. W analogiczny sposób (bramki G2819 i G2820) do pamięci dostarczany jest sygnał *CLR_Tx* umożliwiający ponowny odczyt pamięci i retransmisję pakietu danych.

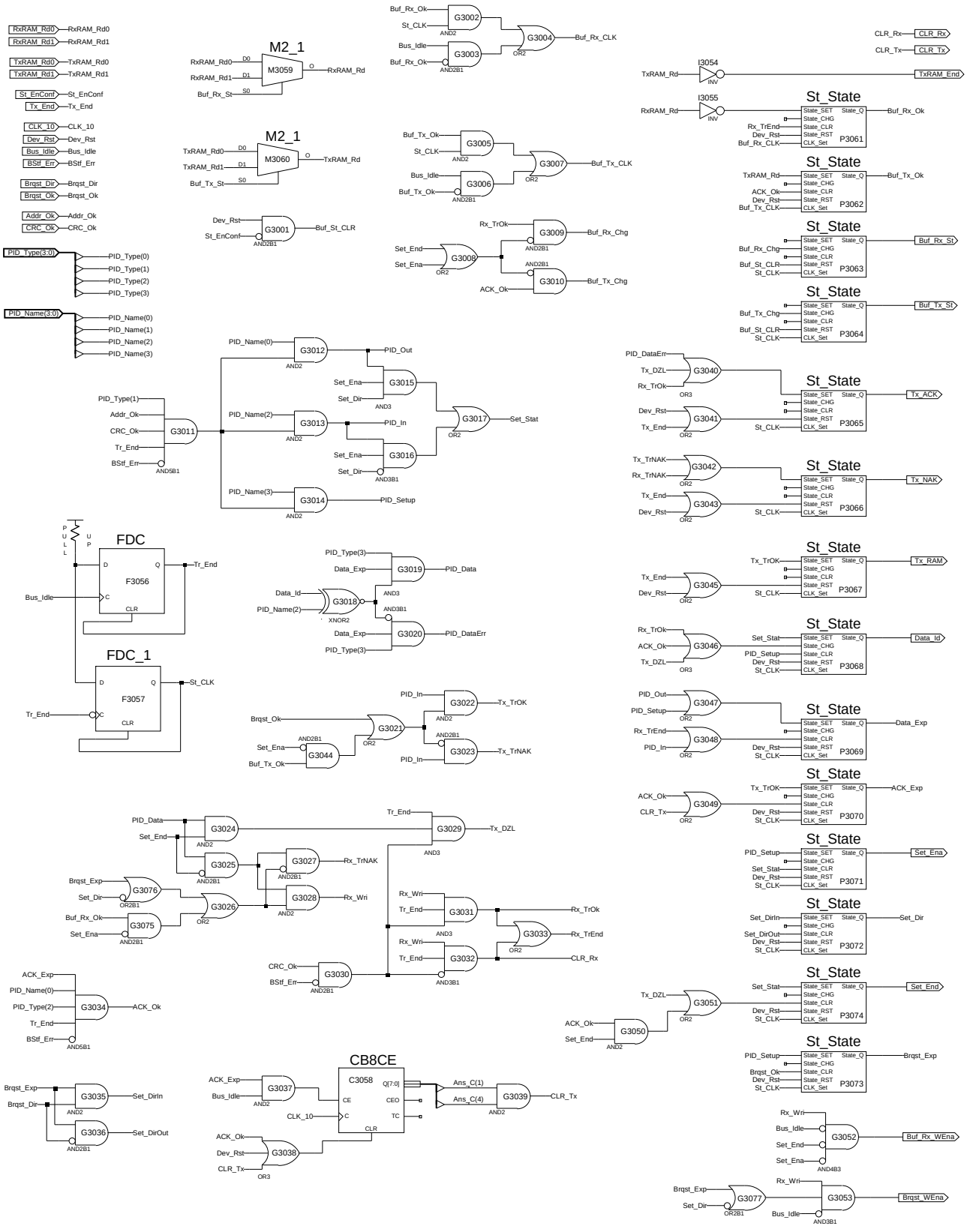
Podobnie odbywa się zapis i odczyt danych ze strony urządzenia, z tą różnicą, że o wyborze danego bloku pamięci decydują układy przerzutników F2826, F2827 i F2828 (bufor wejściowy) oraz F2829, F2830 i F2831 (bufor wyjściowy). Na początkowym etapie konfiguracji (sygnał *St_EnConf* w stanie niskim) każdy sygnał resetu urządzenia *Dev_Rst* powoduje wystąpienie sygnału kasującego zawartość pamięci *RAM_Rst*. Równolegle ten sam sygnał resetuje przerzutniki F2827 i F2830 dołączając urządzenie zewnętrzne do tych samych bloków pamięci co interfejs. Stos po zapisaniu pierwszego bloku P2836 przełącza się na drugi, równocześnie w stan wysoki przechodzi linia *RxRAM_Rd0* informując o gotowości bloku do odczytu. Sygnał ten, wybrany stanem niskim linii *Buf_Rx_Out* z przerzutnika F2830, jest przenoszony przez multiplekser M2833 na wyjście *Out_Ena* informując o gotowości odczytu urządzenie zewnętrzne. Równocześnie multiplekser jednobajtowy **Byte_Mux** dołącza magistralę wyjściową bloku pamięci *Rx_Do0(7:0)* do magistrali danych dostarczanych do urządzenia *Data_Out(7:0)*, zaś bramki kluczujące G2817 i G2818 umożliwiają urządzeniu zwiększanie adresu pamięci P2836 sygnałem *Out_CLK* przez linię *Rx_CLK_Rx0*. Po zakończeniu odczytu przejście linii *Out_Ena* w stan niski (związane z opadnięciem sygnału *RxRAM_Rd0*) informuje urządzenie o opróżnieniu bufora oraz powoduje wystąpienie stanu wysokiego na wyjściu *Q* przerzutnika F2829. Najbliższe zbocze narastające sygnału zegarowego *CLK_8* powoduje dzięki inwerterowi I2825 przełączenie przez przerzutnik F2830 stanu linii *Buf_Rx_Out* na przeciwny oraz wytworzenie przez przerzutnik F2831 krótkiego impulsu kasującego przerzutnik F2829. Przełączenie sygnału *Buf_Rx_Out* na przeciwny umożliwia urządzeniu odczyt drugiego bloku pamięci, o ile jest on gotowy (linia *RxRAM_Rd1* w stanie wysokim).

Dane z urządzenia przeznaczone do zapisu w buforze wejściowym interfejsu są dostarczane magistralą *Data_In(7:0)* do obu bloków P2838 i P2839 równocześnie. Wybór bloku odbywa się poprzez sygnał *Buf_Tx_Out* z przerzutnika F2827. Na podstawie tego sygnału multiplekser M2832 dostarcza poprzez inwerter I2823 informację o dostępności danego bloku pamięci do zapisu na linię *In_Ena* (blok jest dostępny, gdy nie jest gotowy do odczytu, czyli odpowiedni sygnał *TxRAM_Rd0* lub *TxRAM_Rd1* jest w stanie niskim, a stąd *In_Ena* w wysokim). Urządzenie informuje o zamiarze zapisu danych wystawiając linię *In_Write* w stan wysoki. O ile bufor jest wolny, to bramka G2810 przekazuje logiczną jedynkę z *In_Write* na linię *Out_TxEnd* oraz, poprzez bramki G2811 i G2812, do wybranego bloku pamięci jako sygnał *Tx_RAM_WEna0* lub *Tx_RAM_WEna1*. Sygnał zegarowy zapisu *In_CLK* generowany przez urządzenie, jest kluczowany przez bramki G2813 i G2814 do odpowiedniego bloku pamięci jako sygnał zapisu danych *Tx_WCLK0* i *Tx_WCLK1*. Poprzez bramki G2815 i G2816 tworzony jest komplementarny sygnał zegarowy zwiększający adres pamięci na zboczu opadającym *In_CLK* (*Tx_AddrCLK0* bądź *Tx_AddrCLK1*). Po zapełnieniu pamięci (sygnał *In_Ena* opada) bądź zasygnalizowaniu końca zapisu ze strony urządzenia (przejście *In_Write* w stan niski) na wyjściu bramki G2810 pojawia się zero logiczne blokujące na bramkach G2811 i G2812 sygnały zezwalające na zapis. Równocześnie opadające zbocze linii *Out_TxEnd* powoduje wystawienie stanu wysokiego na wyjściu *Q* przerzutnika F2826. Analogicznie do układu bufora wyjściowego, zbocze zegara *CLK_8* przełącza linię *Buf_Tx_Out* w stan przeciwny i kasuje przerzutnik F2826.

7.6 Maszyna stanów skończonych (St_FSM)

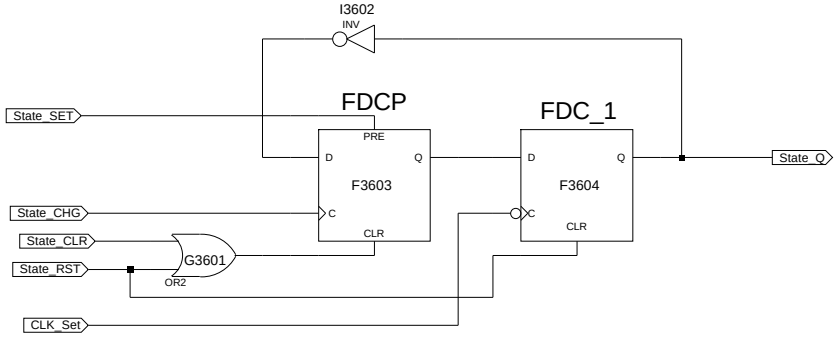
Na rysunku (45) przedstawiono schemat ideowy maszyny stanów skończonych stanowiącej główny blok zarządzający pracą pozostałych części stosu USB. Jak wyjaśniono w podrozdziale (2.9),

znaczenie pakietu o danym nagłówku jest uzależnione od wcześniej odebranych pakietów. Stąd też działania podjęte w wyniku odebrania danego pakietu nie mogą być determinowane czystą



Rysunek 45: Maszyna stanów skończonych - schemat ideowy (**St_FSM**).

logiką kombinacyjną interpretującą rodzaj odebranego nagłówka, lecz muszą uzależnione od wcześniejszych wyników tej logiki (nagłówków odebranych wcześniej). Wynika stąd konieczność zapisu i przechowywania odpowiedzi na dany typ nagłówka w sposób zapewniający ich odczyt przy odbiorze następnych pakietów, a zarazem w sposób uniemożliwiający zmianę stanu sygnału, stanowiącego zarazem wejście logiki kombinacyjnej, jej odpowiedzią na aktualnie odebrany nagłówek. W tym celu zaprojektowano dwuetapowy zapis stanów logicznych maszyny sterowany sygnałami generowanymi przez przerzutniki F3056 i F3067. Stany przechowywane są w blokach **St_State**, których schemat ideowy przedstawiono na rysunku (46).



Rysunek 46: Schemat ideowy bloku przechowującego stan maszyny FSM (**St_State**).

Wraz z zakończeniem odbioru pakietu, Transceiver ustawia linię *Bus_Idle* w stan wysoki, co powoduje krótkotrwałe wystawienie przez przerzutnik F3056 stanu wysokiego na linii *Tr_End*. Sygnał ten jest wykorzystywany przez logikę kombinacyjną maszyny do wygenerowania odpowiedzi na zewnętrzne sygnały sterujące (typ nagłówka, poprawność adresu, itp.), która jest podawana na jedno z wejść *State_SET*, *State_CHG* lub *State_CLR* odpowiedniego bloku **St_State** w zależności od wymaganej w danej sytuacji zmiany stanu. Sygnał *State_SET* powoduje ustawienie wyjścia *Q* przerzutnika F3603 w stan wysoki, sygnał *State_CLR*, poprzez bramkę G3601 - ustawienie stanu niskiego na wyjściu *Q*, zaś sygnał *State_CHG* umożliwia przepisanie na wyjście *Q* przerzutnika F3603 zanegowanego przez inwerter I3602 aktualnego stanu z linii *State_Q*.

Stan wysoki na linii *Tr_End* powoduje równocześnie reset przerzutnika F3056 i pojawienie się na jego wyjściu *Q* stanu niskiego. To przejście powoduje powstanie krótkiego dodatniego impulsu na wyjściu *Q* przerzutnika F3057. Poprzez linię *St_CLK* sygnał ten jest dostarczany do wejścia *CLK_Set* bloku **St_State** powodując przepisanie przez przerzutnik F3604 na zboczu opadającym ustawionego wcześniej na wyjściu przerzutnika F3603 stanu na linię *State_Q*. Dodatkowo blok **St_State** wyposażony został w wejście *State_RST* umożliwiające asynchroniczne skasowanie zawartości obu przerzutników. W dalszej części pracy poprzez terminy "ustawienie linii stanu", "przełączenie linii stanu" oraz "skasowanie/zerowanie linii stanu" rozumiany będzie cały wyżej opisany proces, począwszy od podania logicznej jedynki na odpowiednie wejście bloku **St_Stack**, aż do zapisania zadanej wartości na jego wyjściu.

W tabeli 4 zawarto krótki opis znaczenia poszczególnych sygnałów wyjściowych bloków stanów **St_State**. Bloki P3061-3064 zajmują się przechowywaniem stanów związanych z obsługą bloków pamięci bufora wejścia - wyjścia **St_Buff**. Multiplekser M3059 przenosi na linię *RxRAM_Rd* informację o dostępności wybranego sygnałem *Buf_Rx_St* bloku bufora wyjściowego. Niski sygnał z linii *RxRAM_Rd* (pamięć dostępna do zapisu) ustawia przez inwerter I3055 jedynkę logiczną na linii stanu *Buf_Rx_Ok* podczas narastającego zbocza sygnału *Buf_Rx_CLK*, generowanego przez układ bramek G3002, G3003 i G3004. Jeżeli bufor wyjściowy jest dostępny

Nazwa sygnału	Numer bloku	Opis
Buf_Rx_Ok	P3061	Wybrany blok pamięci bufora wyjściowego jest gotowy do zapisu
Buf_Tx_Ok	P3062	Wybrany blok pamięci bufora wejściowego jest gotowy do odczytu
Buf_Rx_St	P3063	Numer bloku pamięci bufora wyjściowego
Buf_Tx_St	P3064	Numer bloku pamięci bufora wejściowego
Tx_ACK	P3065	Należy nadać pakiet 'Handshake ACK'
Tx_NAK	P3066	Należy nadać pakiet 'Handshake NAK'
Tx_RAM	P3067	Należy nadać pakiet 'Data'
Data_Id	P3068	Identyfikator następnego pakietu 'Data' ('DATA0'/'DATA1')
Data_Exp	P3069	W następnej kolejności spodziewany jest odbiór pakietu 'Data'
ACK_Exp	P3070	Spodziewany jest odbiór 'Handshake ACK' na nadany pakiet 'Data'
Set_Ena	P3071	Wszystkie pakiety 'Data' to dane konfiguracyjne
Set_Dir	P3072	Kierunek transmisji konfiguracyjnych (0-do, 1-z interfejsu)
Brqst_Exp	P3073	Spodziewany pakiet 'Data' zawierający rozkazy konfiguracyjne
Set_End	P3074	Procedura kończąca fazę konfiguracji (pakiet 'Data zero length')

Tabela 4: Opis sygnałów stanów maszyny FSM.

(*Buf_Rx_Ok* w stanie wysokim), to po każdym odebranym pakiecie (dodatnia szpilka na linii *St_CLK*) generowany jest sygnał zegarowy (G3002) przepisujący aktualny stan dostępności bufora. Ponieważ zmiana sygnału selekcji bloku pamięci bufora *Buf_Rx_St* następuje na zboczu sygnału *St_CLK*, więc na linię *Buf_Rx_Ok* przepisany zostanie stan poprzedniego bloku pamięci sprzed przełączenia. Dodatkowo, występujący po zakończeniu odbioru każdego pakietu danych sygnał *Rx_TrEnd* zerujący stan *Buf_Rx_Ok* ma priorytet nad ustawiającym *RxRAM_Rd*, wymusi więc przejście sygnału *Buf_Rx_Ok* w stan niski. Przejście to jest wykorzystywane przez bramkę G3003 do generacji dodatkowego sygnału zegarowego zapisującego uaktualniony po przełączeniu bloków pamięci stan dostępności bufora wyjściowego. Analogicznie działa wybór (*Buf_Tx_St*) i sprawdzanie gotowości (*Buf_Tx_Ok*) bufora wejściowego. Różnicą jest bezpośrednie podanie sygnału *TxRAM_Rd* w wyjścia multipleksera M3060 na wejście *State_SET* bloku P3062, wynikające z odwrotnej interpretacji wysokiego stanu sygnału gotowości bloku pamięci bufora wyjściowego (oznacza on, że pamięć jest zapisana, czyli gotowa do odczytu/nadania pakietu danych). Sygnał zegarowy zapisu stanu *Buf_Tx_CLK* jest tworzony przez bramki G3005, G3006 i G3007 na takiej samej zasadzie, jak opisany wyżej *Buf_Rx_CLK*. Dodatkowo zanegowany inwerterem I3054 sygnał *TxRAM_Rd*, podawany na linię *TxRAM_End*, sygnalizuje stanem wysokim zakończenie odczytu danych z bufora wejściowego.

Ponieważ wystąpienie sygnału resetu magistrali (sygnalizowane stanem wysokim *Dev_Rst*) jest równoznaczne z przerwaniem wszystkich trwających protokołów transmisji, wykorzystano ten sygnał do zresetowania stanów przechowywanych w blokach stanu. Linia *Dev_Rst* podpięta jest do wejść *State_RST* bloków bezpośrednio, bądź poprzez bramki G3041, G3043 i G3045 w przypadku bloków P3065, P3066 i P3067, wymagających asynchronicznego resetu dodatkowym sygnałem (co opisano niżej). Wyjątkiem są bloki P3063 i P3064, przechowujące numer dołączonych do interfejsu bloków pamięci. Kasowanie ich zawartości każdym sygnałem resetu magistrali mogłoby doprowadzić do wspomnianej wcześniej sytuacji kolizji dostępu z urządze-

niem zewnętrznym. Dlatego też resetowanie tych bloków do stanu podstawowego sygnałem *Buf_St_CLR* jest możliwe jedynie do momentu zakończenia procesu enumeracji[16]. Takie działanie zapewnia bramka G3001, blokując sygnały resetu *Dev_Rst* po przejściu linii *St_EnConf* w stan wysoki.

Analiza protokołów transmisji rozpoczyna się od rozpoznania pakietów typu 'Token', sygnalizowanych przez blok **St_PID** stanem wysokim linii *PID_Type(1)*. Po zakończeniu odbioru pakietu, sygnalizowanego stanem wysokim sygnału *Tr_End*, bramka G3011 dokonuje sprawdzenia poprawności adresu *Addr_Ok*, sumy kontrolnej *CRC_Ok* oraz braku błędu "Bit-Stuffingu" (linia *BStf_Err* w stanie niskim). Stan wysoki na wyjściu bramki sygnalizuje poprawny odbiór pakietu typu 'Token', zezwalając na analizę podtypu bramkom G3012 (podtyp 'OUT'), G3013 ('IN') i G3014 ('SETUP'). Jedynka logiczna na odpowiedniej linii magistrali *PID_Name(3:0)* (zgodnie z tabelą 3, s. 5. pracy) powoduje wystawienie w stan wysoki jednej z linii *PID_Out*, *PID_In* lub *PID_Setup*. Poprawny odbiór pakietu 'Token OUT' powoduje ustawienie linii stanu *Data_Exp*, oznaczającej nadejście, jako następnego, pakietu zawierającego dane. W przypadku odbioru pakietu 'Token IN', maszyna sprawdza stan gotowości bufora wyjściowego (bramki G3021, G3022, G3023 i G3044) oraz, dla zabezpieczenia przed błędem transmisji, kasuje linię stanu *Data_Exp*. Podczas protokołu wymiany danych (linie *Set_Ena* i *Brqst_Ok* w stanie niskim) sygnał *Buf_Tx_Ok* jest przekazywany przez bramki G3044 i G3021 bez zmian. Jeśli bufor jest gotowy do nadania danych (stan wysoki), poprzez bramkę G3022 ustawiana jest logiczna jedynka na linii *Tx_TrOK*. W wyniku tego ustawiane są linie stanu *Tx_RAM* (oznaczająca rozkaz generacji pakietu danych dla nadajnika) oraz *Ack_Exp* (w wyniku nadania pakietu 'Data' spodziewany jest protokół potwierdzenia). Gdy bufor jest pusty, w stanie wysokim znajduje się linia *Tx_TrNAK* powodując ustawienie linii stanu *Tr_NAK* oznaczającej rozkaz nadania pakietu 'Handshake NAK'. Wyjątek stanowi trwający protokół konfiguracyjny (linia *Set_Ena* w stanie wysokim). Wówczas bramka G3044 blokuje sygnał gotowości bufora wejściowego, zaś nadanie pakietu 'Data', zawierającego opis interfejsu, jest uzależnione od poprawnego rozpoznania rozkazów konfiguracyjnych (linia *Brqst_Ok* w stanie wysokim gdy rozkaz został zaakceptowany).

Rozpoznanie poprzez bramkę G3014 pakietu 'Token SETUP' oznacza rozpoczęcie protokołu konfiguracji. Wysoki stan na linii *PID_Setup* powoduje ustawienie linii stanów *Set_Ena* (protokół konfiguracji w toku), *Brqst_Exp* (następny pakiet 'Data' zawiera rozkazy konfiguracyjne) oraz, poprzez G3047, linii *Data_Exp* umożliwiającej odbiór pakietu danych. Dodatkowo, zgodnie z opisem zawartym w podrozdziale (2.9), wysoki stan linii *PID_Setup* kasuje linię identyfikatora podtypu pakietu 'Data', *Data_Id*. Otrzymanie w protokole konfiguracji (*Set_Ena* w stanie wysokim) pakietu 'Token', o kierunku przeciwnym do zadeklarowanego w rozkazie, oznacza procedurę kończącą ten protokół. Sytuacja ta jest rozpoznawana przez bramki G3015 i G3016 na podstawie sygnałów *Set_Ena* i *Set_Dir*, przechowującego kierunek transmisji konfiguracyjnej. Wystąpienie stanu wysokiego na wyjściu którejkolwiek z bramek jest przekazywane przez G3017 na linię *Set_Stat* powodującą skasowanie linii stanu *Set_Ena*, ustawienie linii *Set_End*, będącej sygnałem dla nadajnika o konieczności wygenerowania pakietu 'Data zero length', oraz ustawienie identyfikatora podtypu pakietu 'Data' na 1 (*Data_Id*).

Identyfikację pakietu danych 'Data' umożliwiają bramki G3019 i G3020. Stan wysoki na linii magistrali *PID_Type(3)*, idcentyfikujący ten pakiet, jest akceptowany tylko w przypadku otrzymania wcześniej pakietu 'Token OUT' o zgodnym adresie (*Data_Exp* w stanie wysokim). Bramka G3018 umożliwia kontrolę zgodności podtypu pakietu z wartością przechowywaną przez stan *Data_Id*. Linia *PID_Name(2)* w stanie wysokim oznacza odbiór pakietu 'DATA1'. Negacja alternatywy wykluczającej tej linii z *Data_Id* da w wyniku stan wysoki tylko wtedy, gdy spodziewanym identyfikatorem pakietu będzie 1. W przypadku niezgodności oczekiwanego i odebranego podtypu w stan wysoki ustawiana jest linia *PID_DataErr* dzięki czemu, przez bramkę G3040,

[16]Universal Serial Bus Specification, rev. 2.0, s. 241

ustawiana jest linia stanu *Tx_ACK* oznaczająca rozkaz generacji pakietu 'Handshake ACK'. Zgodność identyfikatorów jest sygnalizowana stanem wysokim linii *PID_Data*, co umożliwia analizę stanu dostępności bufora wyjściowego przez bramki G3027 i G3028 pod warunkiem, że nie jest to pakiet 'Data zero length' (stan wysoki na wyjściu bramki G3025). Bramka G3075 blokuje sygnał dostępności bufora *Buf_Rx_Ok* podczas protokołu konfiguracji, zaś bramka G3076 umożliwia wówczas odbiór danych konfiguracyjnych do bloku **St_Brqst**, o ile spodziewany jest blok rozkazów *Brqst_Exp* lub zadeklarowana została transmisja konfiguracji ze strony komputera (*Set_Dir* w stanie niskim). Stan wysoki na wyjściu bramki G3021, oznaczający gotowość do odbioru danych, umożliwia bramce G3022 wystawienie jedynki logicznej na linii *Rx_Wri*. Odrzucenie pakietu danych następuje poprzez ustawienie linii stanu *Tx_NAK* wysokim sygnałem przekazywanym przez linię *Rx_TrNAK* i bramkę G3042. Stan wysoki linii *Rx_Wri* jest wykorzystywany przez bramki G3052 i G3053 do wytworzenia sygnałów zezwalających na zapis danych do bufora (*Buf_Rx_WEna*) lub bloku **St_Brqst** (*Brqst_WEna*). Pierwszy z sygnałów jest blokowany podczas protokołu konfiguracji wysokimi sygnałami linii *Set_Ena* i *Set_End*. Wystawienie drugiego z nich jest uzależnione od poprawnego kierunku transmisji danych (*Set_Dir* w stanie niskim) lub oczekiwania na rozkazy konfiguracyjne *Brqst_Exp*. Oba sygnały zezwalające są utrzymywane do momentu zakończenia odbioru pakietu danych, sygnalizowanego przejściem linii *Bus_Idle* w stan wysoki.

W przypadku procedury kończącej protokół konfiguracji (jedynka logiczna na linii *Set_End*) stan wysoki na wyjściu bramki G3024 (a, zarazem, niski na G3025) pozwala na pominięcie etapu odczytu danych z pakietu. Bramka G3030 sygnalizuje poprawność transmisji (brak błędu "Bit-Stuffingu" *BStf_Err* i poprawność sumy kontrolnej *CRC_Ok*). Sygnał ten jest analizowany przez bramki G3029, G3031 i G3032 po zakończeniu odbioru pakietu, co zapewnia linia *Tr_End*. Na tej podstawie, przy sygnale pominięcia odczytu danych, bramka G3029 ustawia poprzez sygnał *Tx_DZL* i bramkę G3040 linię stanu *Tx_ACK* umożliwiając tym samym potwierdzenie odbioru 'Data zero length'. Równocześnie, z pomocą bramki G3046, następuje zmiana stanu *Data_Id* (jako że odebrano pakiet danych w sposób poprawny) oraz skasowanie przez bramkę G3051 linii stanu *Set_End*, co ostatecznie kończy protokół konfiguracji. Przy protokole odbioru danych poprzez bramkę G3031, sygnał *Rx_TrOk* i bramkę G3040 następuje ustawienie linii stanu *Tr_ACK* oraz zmiana przez bramkę G3046 stanu linii *Data_Id* na przeciwny. W przypadku wystąpienia błędu transmisji, za pomocą bramki G3032 linia *CLR_Rx* zostaje ustawiona w stan wysoki. Stanowi ona sygnał dla bufora wyjściowego **St_Buff**, że odebrane dane są niepoprawne i należy je usunąć poprzez skasowanie licznika adresowego. Zarazem brak rozkazu generacji pakietu potwierdzenia 'Hadshake' informuje komputer o konieczności retransmisji pakietu 'Data' o tym samym identyfikatorze, co odrzucony. Z tego też powodu stan linii *Data_Id* nie zostaje zmieniony. Niezależnie od powodzenia odbioru danych, linia stanu *Data_Exp* zostaje skasowana poprzez bramkę G3033, sygnał *Rx_TrEnd* i bramkę G3048.

Po nadaniu pakietu 'Data' interfejs oczekuje na potwierdzenie ze strony komputera pakietem 'Handshake ACK', co sygnalizowane jest stanem wysokim linii *ACK_Exp*. Jeżeli pakiet nie zostanie otrzymany po upływie więcej niż 16 taktów zegara od momentu zakończenia transmisji pakietu danych[17], oznacza to błąd danych i konieczność retransmisji pakietu. Wykrycie takiego zdarzenia umożliwia licznik C3058 zliczający kolejne takty sygnału zegarowego *CLK_10*. Sygnał zezwalający *CE* licznika jest wystawiany przez bramkę G3037 na podstawie stanu oczekiwania na potwierdzenie *Ack_Exp* w czasie, gdy magistrala USB pozostaje w stanie spoczynku (linia *Bus_Idle* w stanie wysokim). Ponieważ stan *Ack_Exp* jest ustawiany równocześnie z rozkazem nadawania pakietu *Tx_RAM*, licznik C3058 zlicza również dwa takty przerwy między odbiorem pakietu 'Token IN', a rozpoczęciem nadawania przez Transceiverpakietu 'Data'. Dlatego też zwiększono do 18 ilość zliczeń, po których transmisja zostaje uznana za błędną. Liczba

[17]Universal Serial Bus Specification, rev. 2.0, rozdz. 7.1.19.1, s. 168

ta jest dekodowana przez bramkę G3039 na podstawie stanu wysokiego drugiego *Ans_C(1)* i piątego bitu *Ans_C(4)* magistrali wyjściowej licznika C3058. Wystawienie logicznej jedynki na linię *CLR_Tx* skutkuje skasowaniem linii stanu *ACK_Exp* oraz wskazań licznika C3058 przez bramkę G3038. Dodatkowo sygnał *CLR_Tx* informuje bufor wejściowy *St_Buff* o konieczności przywrócenia odczytanego uprzednio bloku pamięci do stanu początkowego, co umożliwia ponowną transmisję danych w nim zawartuch. Pozostałymi sygnałami kasującymi licznik są reset magistrali (*Dev_Rst*) i informacja o otrzymaniu pakietu 'Handshake ACK' *ACK_Ok*. Jest ona tworzona przez bramkę G3034 na podstawie otrzymania sygnału poprawnego (*BStf_Err*) zakończenia odbioru (*Tr_End*) oczekiwanego (*ACK_Exp*) pakietu typu 'Handshake' (*PID_Type(2)*) i podtypu 'ACK' (*PID_Name(0)*). Sygnał odbioru potwierdzenia *ACK_Ok* służy ponadto do skasowania przez bramkę G3049 linii stanu *ACK_Exp*, zmiany przez G3046 stanu *Data_Id* na przeciwny oraz, podczas procedury kończącej protokół konfiguracji (*Set_End* w stanie wysokim), do skasowania przez bramki G3050 i G3051 linii stanu *Set_End*.

Do skasowania stanu oczekiwania na rozkazy konfiguracyjne *Brqst_Exp* służy przesyłany przez blok **St_DConf** sygnał *Brqst_Ok*, informujący o poprawnym rozpoznaniu otrzymanego rozkazu. W przeciwnym wypadku stan ten jest kasowany resetem magistrali *Dev_Rst* występującym w wyniku błędu protokołu konfiguracji interfejsu. Kierunek transmisji protokołu *Set_Dir* jest ustawiany poprzez bramki G3035 i G3036 na podstawie sygnału *Brqst_Dir* pochodzącego z odczytania rozkazu przez blok **St_DConf**. W zależności od deklarowanego kierunku (0 - komputer do interfejsu, 1 - na odwrót) wystawiany jest stan wysoki na odpowiedniej linii *Set_DirOut* lub *Set_DirIn*, kasującej lub ustawiającej linię stanu *Set_Dir*. Sygnał *Brqst_Exp* blokuje pracę bramek G3035 i G3056 w sytuacji, gdy w miejsce rozkazów odbierane są inne dane konfiguracyjne.

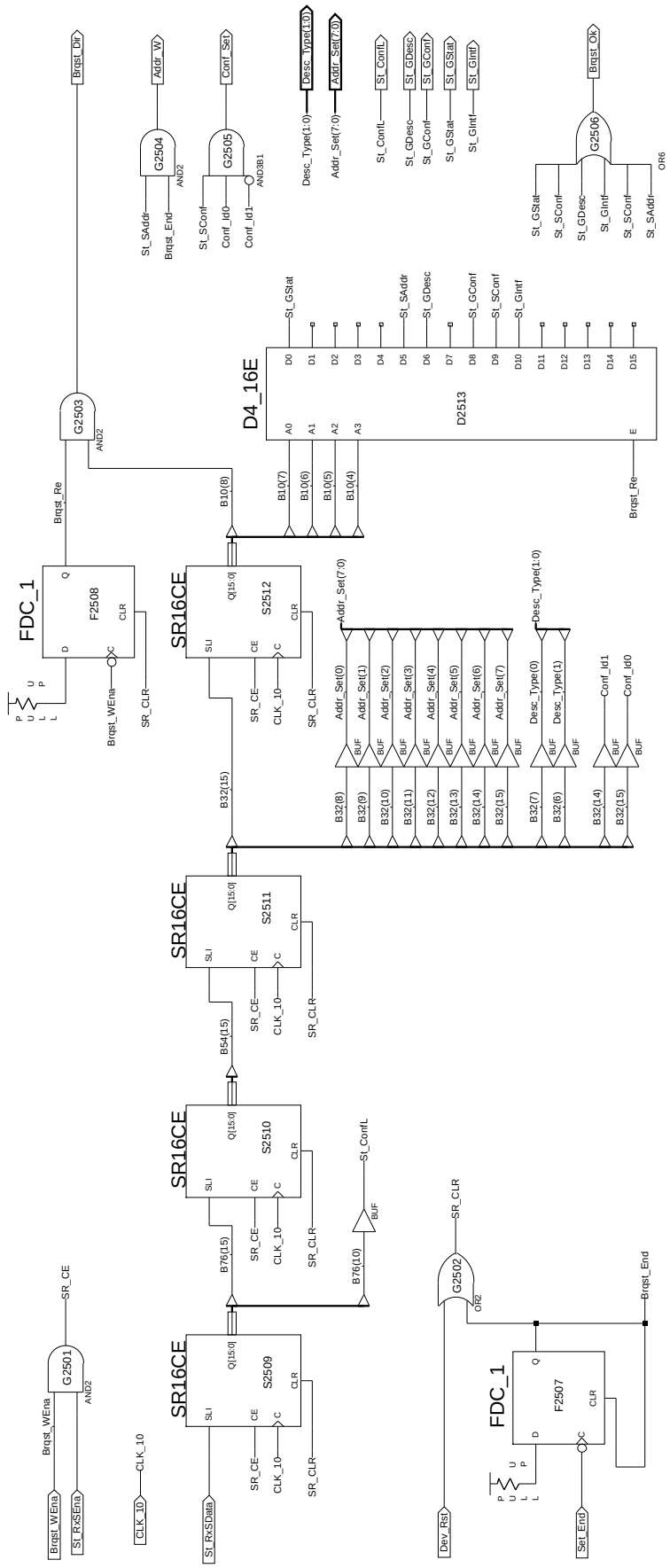
Koniec nadawania pakietu, wymuszonego liniami stanów *Tx_ACK*, *Tx_NAK* lub *Tx_RAM*, jest sygnalizowane przez blok generacji nagłówka **St_Tx** wysokim stanem linii *Tx_End*. Aby blok nadawania nie rozpoczął ponownie pracy, wymienione wyżej linie stanów muszą zostać skasowane natychmiast po otrzymaniu sygnału końca nadawania. Uzyskano to podłączając linię *Tx_End* do wejść resetu asynchronicznego bloków P3065, P3066 i P3067 przez, odpowiednio, bramki G3041, G3043 i G3045. Bramki te umożliwiają dodatkowo reset stanów sygnałem *Dev_Rst* tak, jak przewidziano to dla (prawie) wszystkich bloków **St_Stack**.

7.7 Blok interpretacji rozkazów konfiguracyjnych (St_Brqst)

Na rysunku (47) przedstawiono schemat bloku interpretującego rozkazy konfiguracyjne. Pomimo iż konstrukcja maszyny stanów **St_FSM** przewiduje możliwość odbioru danych w protokole konfiguracji, na tym etapie pracy nie spotkano się z takim przypadkiem, podobnie jak z koniecznością rozpoznania i wykonania niektórych rozkazów protokołu. Blok zaprojektowano jednak w taki sposób, by na dalszym etapie prac (poza ramami pracy inżynierskiej) istniała możliwość jego prostej rozbudowy w celu rozszerzenia zakresu obsługiwanych rozkazów i poleceń.

Ponieważ blok rozkazów nadawanych po pakiecie 'Token SETUP' jest zawsze ośmiobajtowy[18], do jego zapisu zastosowano cztery szesnastobitowe rejestry przesuwne S2509-2512. Taka konstrukcja, poza możliwością łatwej rozbudowy rozmiaru, umożliwia przede wszystkim równoczesny odczyt wszystkich bajtów rozkazu, co byłoby niemożliwe przy zastosowaniu bloku pamięci RAM.

[18]Universal Serial Bus Specification, rev. 2.0, Tabela 9-2, s. 248



Rysunek 47: Schemat ideowy bloku interpretacji rozkazów konfiguracyjnych (**St_Brqst**).

Zawartość rejestrów jest kasowana stanem wysokim linii *SR_CLR* stanowiącej wyjście bramki G2502. Po zakończeniu protokołu konfiguracji linia *Set_End* przechodzi w stan niski, co jest wykrywane przez przerzutnik F2507 generujący krótki impuls dodatni na linii *Brqst_End*. Impuls ten jest przepuszczany przez bramkę G2052 kasując zawartość rejestrów, podobnie jak sygnał resetu magistrali USB *Dev_Rst*. Sygnałem zezwalającym rejestrom S2509-2512 na przepisywanie danych z wejścia szeregowego *St_RxSData* na zboczu sygnału zegarowego *CLK_10* jest, generowany przez maszynę stanów, wysoki stan na linii *Brqst_WEna* połączony przez bramkę G2501 z wysokim stanem zezwalającym na przepisywanie danych *St_RxSEna*, pochodzącym z bloku **St_RxCRC**.

Ze względu na zastosowany sposób połączeń (najstarszy bit rejestru poprzedzającego stanowi wejście następnego), wszystkie cztery rejestry S2509-2512 stanowią de facto jeden duży, 64-bitowy rejestr przesuwny. Z tego względu pierwszy bit odebranych danych (LSB bajtu 'bmRequetType')[19] zajmuje najstarszą, szesnastą komórkę ostatniego w szeregu rejestru S2512 (linia *B10(15)*). Umiejscowienie poszczególnych bajtów bloku rozkazu w rejestrach przedstawiono w tabeli 5.

Nazwa rozkazu	Numer rejestru	Zakres magistrali wyjściowej
'bmRequestType'	S2512	*B10(15:8)*
'bRequest'	S2512	*B10(7:0)*
'wValue', młodszy bajt	S2511	*B32(15:8)*
'wValue', starszy bajt	S2511	*B32(7:0)*
'wIndex', młodszy bajt	S2510	*B54(15:8)*
'wIndex', starszy bajt	S2510	*B54(7:0)*
'wLength', młodszy bajt	S2509	*B76(15:8)*
'wLength', starszy bajt	S2509	*B76(7:0)*

Tabela 5: Zapis bloku rozkazów w rejestrach S2509-2512.

Spośród jedenastu definiowanych przez specyfikację rozkazów[20], blok rozpoznaje na obecnym etapie pięć. Ich których krótki opis przedstawiono w tabeli 6. Początkowo przewidywano jeszcze potrzebę obsługi rozkazu 'GET_INTERFACE' umożliwiającego odczyt konfiguracji dodatkowego interfejsu, którego budowa nie okazała się być jednak potrzebna. Ponieważ alternatywny interfejs nie jest zgłaszany w opisie, rozkaz ten nie przychodzi i linia (*St_GIntf*) jest nieużywana (w dalszym opisie została pominięta).

W tabeli i 7 zamieszczono krótki opis rozpoznawanych i wysyłanych przez blok **St_DConf** typów opisów (deskrytorów)[21]. Typ żądanego przez komputer opisu zawarty jest starszym bajcie 'wValue'. Dwa najmłodsze bity tego bajtu (*B32(7)* i *B32(6)*) zostają pobrane i poprzez magistralę *Desc_Type(1:0)* przesłane do bloku **St_DConf** determinując w ten sposób rodzaj nadawanego opisu.

[19]Universal Serial Bus Specification, rev. 2.0, Tabela 9-2, s. 248

[20]Universal Serial Bus Specification, rev. 2.0, Tabela 9-3 i Tabela 9-4, s. 250-251

[21]Universal Serial Bus Specification, rev. 2.0, Tabela 9-5, s. 251

Nazwa rozkazu	Nazwa linii	Opis
'GET_STATUS'	*St_GStat*	Żądanie wysłania informacji o aktualnym statusie interfejsu bądź jednej z jego składowych.
'SET_ADDRESS'	*St_SAddr*	Nadanie unikalnego adresu interfejsu zamieszczonego w młodszym bajcie 'wValue'
'GET_DESCRIPTOR'	*St_GDesc*	Żądanie wysłania jednego z ośmiu typów opisu wyszczególnionych w tabeli 7
'GET_CONFIGURATION'	*St_GConf*	Żądanie wysłania informacji o ustawionej konfiguracji interfejsu
'SET_CONFIGURATION'	*St_SConf*	Wybór jednej z możliwych konfiguracji interfejsu poprzez podanie jej numeru w młodszym bajcie 'wValue'

Tabela 6: Rozkazy interpretowane przez blok **St_Brqst**.

Rodzaj deskryptora	*Desc_Type(i)* 1	0	Opis
'DEVICE'	0	1	Opis interfejsu zawierający m.in. nazwę i typ, standard USB i ilość możliwych konfiguracji
'CONFIGURATION'	1	0	Opis wszystkich zadeklarowanych konfiguracji zawierający informację o samym interfejsie, jak i wszystkich jego składowych (bufory, itp.)
'STRING'	1	1	Tekst w formacie 'Unicode' wyświetlany w systemie operacyjnym jako opis interfejsu, lub deklaracja języka komunikatów (np. Angielski)

Tabela 7: Typy obsługiwanych deskryptorów.

Po zakończeniu odbioru pakietu rozkazów sygnał *Brqst_WEna* przechodzi w stan niski, co jest wykrywane przez przerzutnik F2508, który w wyniku tego ustawia linię *Brqst_Re* w stan wysoki. Informację o deklarowanym kierunku transmisji zawiera MSB bajtu 'bmRequetType', stąd zostaje ona pobrana z linii *B10(8)* rejestru S2512 i przekazana przez bramkę G2503, po zakończeniu odbioru danych, do maszyny stanów linią *Brqst_Dir*. Oprócz tego, wysoki stan na linii *Brqst_Re* umożliwia pracę dekoderowi "4 do 16" D2513 zajmującemu się rozpoznawaniem rozkazów zawartych w, drugim z kolei, bajcie 'bRequest'. Ponieważ do zakodowania jedenastu rozkazów wystarczają cztery (młodsze) bity, tylko one są podpięte do wejść dekodera (linie *B10(7)-B10(4)*). Jeżeli rozkaz jest obsługiwany przez interfejs, stan wysoki jednej z wymienionych w tabeli 6 linii jest przenoszony przez bramkę G2506 na linię *Brqst_Ok*, informując maszynę stanów o rozpoznaniu rozkazu. Informacja o otrzymaniu rozkazu zawierającego żądanie wysłania danych konfiguracyjnych jest przesyłana do bloku **St_DConf** poprzez jedną z linii

St_GDesc, *St_GConf* lub *St_GStat*. Stan wysoki linii *Brqst_Re* jest utrzymywany do momentu wystawienia przez bramkę G2502 sygnału *SR_CLR* kasującego przerzutnik F2508.

W przypadku opisu konfiguracji interfejsu (rozkaz 'GET_DESCRIPTOR' o rodzaju 'CONFIGURATION') występują dwie możliwości - wysłania krótkiego, dziewięciobajtowego opisu zawierającego wyłącznie podstawowe informacje, lub długiego, 32-bajtowego z pełnym opisem interfejsu i jego składowych. Standardowo komputer żąda opisu krótkiego, deklarując jego długość w młodszym bicie bajtu 'wLength'. Po jego otrzymaniu ponawia żądanie, tym razem jako długość podając wartość zadeklarowaną na końcu krótkiego deskryptora (32 bajty w przypadku omawianego interfejsu). Rozpoznanie żądanej długości umożliwia sygnał *St_ConfL* pobierający wartość z jedenastej komórki rejestru S2509, czyli szóstemu bitowi młodszego bajtu 'wLegth' (32 odpowiada liczbie binarnej 00100000B). Stąd stan wysoki tej linii informuje blok **St_DConf** o konieczności przesłania długiego opisu konfiguracji.

Linie *Conf_Id0* oraz *Conf_Id1* wraz z bramką G2505 odpowiadają za poprawne rozpoznanie konfiguracji nadawanej rozkazem 'SET_CONFIGURATION'. Numer wybranej przez komputer konfiguracji jest przesyłana w młodszym bajcie 'wValue'. Ponieważ interfejs posiada tylko jedną konfigurację, ona też powinna zostać wybrana poprzez przesłanie w tym bajcie wartości 00000001B. Stąd linia *Conf_Id0* powinna znajdować się w stanie wysokim (LSB młodszego bajtu 'wValue'), zaś linia *Conf_Id1* w niskim. Dekodowaniem takiego stanu zajmuje się bramka G2505, co w połączeniu ze stanem wysokim linii *St_SConf* oznacza poprawne ustawienie konfiguracji numer 1, a, co za tym idzie, poinformowanie poprzez linię *Conf_Set* bloku **St_Enum** o przejściu interfejsu w stan skonfigurowania.

Unikalny adres interfejsu, nadawany rozkazem 'SET_ADDRESS', zostaje przesłany przez komputer w młodszym bajcie 'wValue'. Jest stamtąd pobierany (linie *B32(15)-B32(8)* magistrali wyjściowej rejestru S2511) i przekazywany poprzez magistralę *Addr_Set(7:0)* do bloku kontroli poprawności adresu **St_Addr**, gdzie zostaje zapisany na narastającym zboczu sygnału *Addr_W*. Sygnał ten jest tworzony poprzez przepuszczenie przez bramkę G2504 krótkiego impulsu *Brqst_End* występującego na końcu protokołu konfiguracji. Warunkiem utworzenia sygnału *Addr_W* jest wysoki stan linii *St_SAddr* identyfikujący rozkaz nadania adresu. Pomimo że adres interfejsu jest siedmiobitowy, pobierany jest dodatkowo ósmy bit w celu uzgodnienia rozmiarów magistrali *Addr_Set(7:0)* i wejścia *D[7:0]* rejestru S2002 w bloku **St_Addr**. Jak wcześniej napisano[22], bit ten jest pomijany.

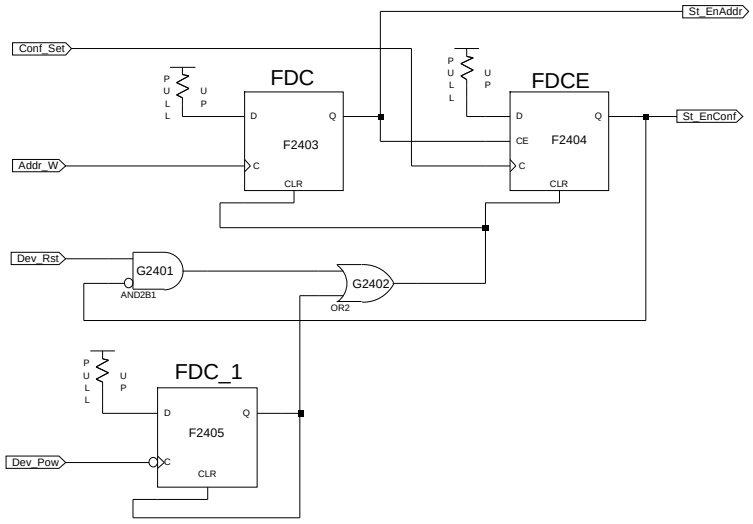
7.8 Blok stanu enumeracji interfejsu (St_Enum)

Na rysunku (48) przedstawiono schemat ideowy bloku zapisującego i przechowującego stan enumeracji interfejsu.

Ponieważ w trakcie projektowania uznano, że informacja o pierwszym etapie enumeracji, czyli stanie zasilania, nie jest potrzebna do pracy interfejsu, stan ten nie jest przechowywany. Sygnał podpięcia do magistrali (czyli przejście w stan niski linii *Dev_Pow*) służy jedynie do skasowania wyższych stanów, które mogły zostać ustawione w sytuacji, gdy interfejs był już wcześniej skomunikowany z komputerem. Opadające zbocze sygnału *Dev_Pow* powoduje wystawienie przez przerzutnik F2405 krótkiej szpilki na wyjściu *Q*, skąd zostaje ona przekazana przez bramkę G2402 na wejścia kasujące *CLR* przerzutników F2403 i F2404. Na drugie wejście bramki G2402 podany jest sygnał resetu magistrali *Dev_Rst* przepuszczany przez bramkę G2401 do chwili przejścia interfejsu w stan skonfigurowania (linia *St_EnConf* w stanie wysokim).

Przejście w stan zaadresowania następuje poprzez wykrycie przez przerzutnik F2403 zbocza narastającego na sygnale *Addr_W*, zapisującym równocześnie nadany adres w bloku **St_Addr**.

[22]Podrozdział 7.3 niniejszej pracy



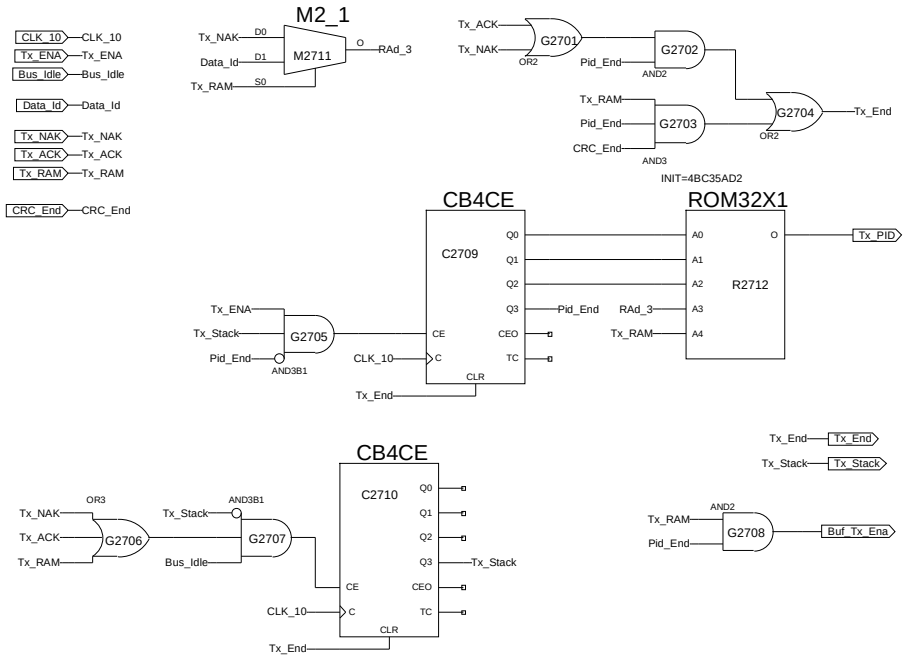
Rysunek 48: Stan enumeracji interfejsu - schemat ideowy (**St_Enum**).

W wyniku tego na linii *St_EnAddr* zostaje ustawiona logiczna jedynka identyfikując stan zaadresowania interfejsu oraz zezwalając na pracę przerzutnika F2404 (wejście *CE*).

Sygnał o poprawnym ustawieniu konfiguracji interfejsu *Conf_Set*, generowany przez blok **St_Brqst**, powoduje wystawienie przez przerzutnik stanu wysokiego na linii *St_EnConf*. Oznacza to przejście interfejsu w stan skonfigurowania i zakończenie procesu enumeracji USB.

7.9 Blok tworzenia nagłówków pakietów (St_Tx)

Rysunek (49) przedstawia schemat ideowy pierwszego z bloków odpowiedzialnych za generację nadawanych pakietów - blok tworzenia nagłówka.



Rysunek 49: Tworzenie nagłówka nadawanych pakietów - schemat ideowy (**St_Tx**).

Sygnałem do generacja pakietu jest wystawienie przez maszynę stanów **St_FSM** logicznej jedynki na jednej z linii *Tx_ACK*, *Tx_NAK* lub *Tx_RAM*. . Stan wysoki którejkolwiek z w/w linii rozpoczynających nadawanie przenoszony jest przez bramkę G2706 na wejście bramki G2707. Jeśli magistrala jest w stanie spoczynku (stan wysoki *Bus_Idle*) oraz nie rozpoczęto jeszcze nadawania pakietu (*Tx_Stack* w stanie niskim), bramka G2707 wystawia sygnał zezwalający licznikowi C2710 na zliczanie kolejnych taktów sygnału zegarowego *CLK_10*. Ponieważ interfejs musi zapewnić przerwę o długości od 3 do 7 taktów zegara pomiędzy odbieranym a nadawanym pakietem[23], zaś po ustawieniu sygnału *Bus_Idle* w stan wysoki Transceiver odbiera jeszcze dwa bity sygnału SE0, zdecydowano się zliczać osiem taktów zegara *CLK_10* przed rozpoczęciem nadawania (2 bity SE0 plus sześć taktów przerwy). Po ósmym zboczu zegara licznik C2710 ustawia w stan wysoki linię *Tx_Stack* blokując samemu sobie możliwość dalszej pracy oraz informując Transceiver o konieczności rozpoczęcia generacji preambuły. Z chwilą zakończenia generacji Transceiver wystawia linię *Tx_ENA* w stan wysoki informując o gotowości do pobierania wysyłanych przez stos danych.

Nagłówki pakietów zostały zapisane w pamięci ROM R2712 o rozmiarze 32 słów jednobitowych. Odczyt pamięci, podobnie jak pamięci RAM zastosowanej w bloku **St_RAM64**, polega na podaniu na wejścia *A0-A4* adresu bitu, który wystawiany jest na wyjściu *O*. Pamięć podzielono na cztery banki zawierające cztery ośmiobitowe nagłówki. Wybór właściwego banku polega na podaniu na dwa najstarsze bity adresowe *A3* i *A4* pamięci odpowiedniej kombinacji sygnałów sterujących. Do wejścia *A4* podpięto bezpośrednio sygnał *Tx_RAM*, zaś do *A3* linię *RAd_3* będącą wyjściem multipleksera M2711. Do jego wejść dołączono sygnały *Tx_NAK* oraz identyfikator pakietu data *Data_Id*, zaś do linii wyboru ponownie *Tx_RAM*, uzyskując łącznie działanie przedstawione w tablicy prawdy 8.

Wejścia multipleksera M2711			Adres banku		Zawartość	Typ i podtyp
Tx_NAK	*Data_Id*	*Tx_RAM*	*A4*	*A3*	banku	nagłówka
0	X	0	0	0	D2H = 11010010B	'Handshake ACK'
1	X	0	0	1	5AH = 01011010B	'Handshake NAK'
X	0	1	1	0	C3H = 11000011B	'Data DATA0'
X	1	1	1	1	4BH = 01001011B	'Data DATA1'

Tabela 8: Tablica prawdy wyboru banku pamięci.

Odczyt i przesłanie do Transceiver-a wybranego w ten sposób nagłówka polega na zaadresowaniu kolejnych ośmiu bitów banku przez podanie na wejścia adresowe *A0-A3* pamięci R2712 wartości od 000B do 111B (zero do siedem dziesiętnie). Adresowaniem pamięci zajmuje się licznik C2709. Sygnał zezwalający na zwiększanie adresu sygnałem zegarowym *CLK_10* pobierany jest z wyjścia bramki G2708. Wystawia ona logiczną jedynkę jeśli rozpoczęto nadawanie pakietu (stan wysoki *Tx_Stack*), Transceiver zasygnalizował pobieranie danych (*Tx_ENA*), a generacja nagłówka nie została jeszcze zakończona (*Pid_End* w stanie niskim). Na początku nadawania na wszystkich wyjściach licznika znajdują się zera logiczne i pamięć R2712 wystawia na linię *Tx_PID* pierwszy bit wybranego nagłówka. Sygnał *Tx_PID* przekazany jest przez blok **St_TxMux** do Transceiver-a, który pobiera jego wartość na sygnale zegarowym *CLK_8*[24]. Dzięki temu, że zwiększenie wartości adresu następuje zegarem *CLK_10*, kolejny bit nagłówka

[23]Universal Serial Bus Specification, rev. 2.0, rozdz. 7.1.18.1, s. 168

[24]Podrozdział 5.5 niniejszej pracy

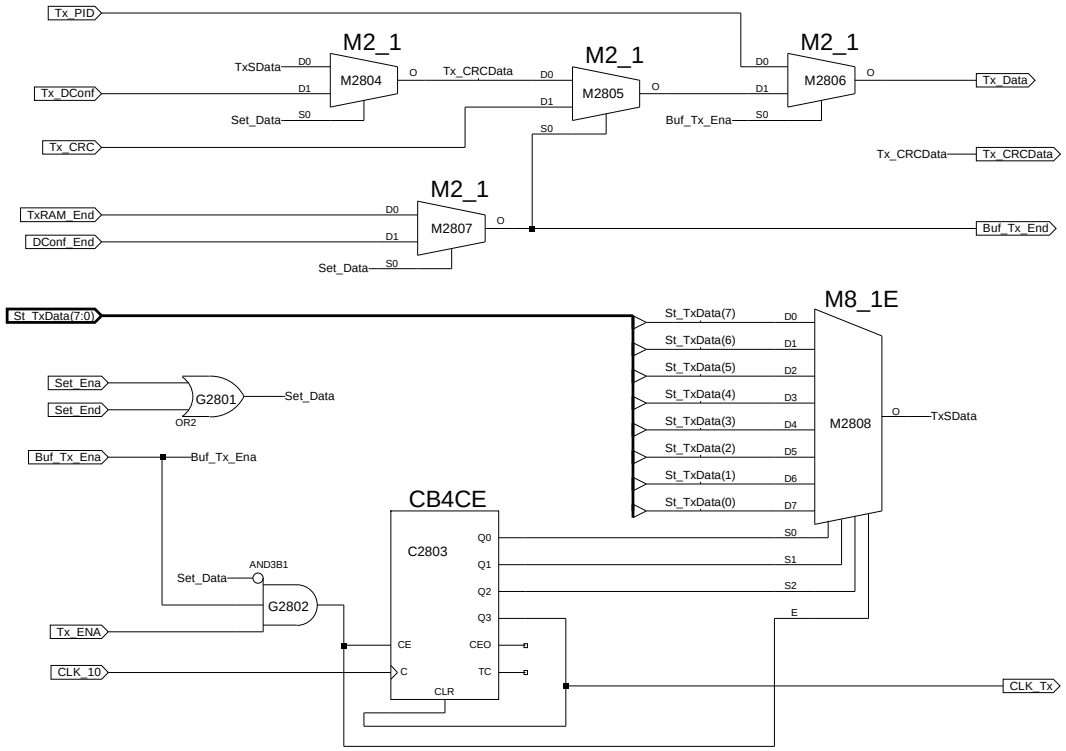
jest wystawiany przez ROM R2712 już po odczycie poprzedniego przez Transceiver. Osiągnięcie przez licznik C2709 wartości 1000B oznacza zakończenie generacji nagłówka, co jest sygnalizowane stanem wysokim linii *Pid_End*.

Ponieważ pakiety typu 'Handshake' zawierają tylko nagłówek, wraz z chwilą jego wysłania powinno zakończyć się nadawanie pakietu. Zapewniają to bramki G2702 i G2704, przenosząc stan wysoki sygnału *Pid_End* na linię *Tx_End*, pod warunkiem, że wysyłanym pakietem był 'Handshake' (jedna z linii *Tx_ACK* lub *Tx_NAK* w stanie wysokim), o czym informuje bramka G2701. Stan wysoki linii *Tx_End* powoduje skasowanie zawartości liczników C2709 i C2710, powodując przejście w stan niski linii *Pid_End* oraz *Tx_Stack*, co informuje Transceiver o końcu nadawania. Dodatkowo stan linii *Tx_End* wykorzystywany jest w maszynie stanów **St_FSM** do skasowania wystawionego stanu nadawania.

Jeżeli nadawany jest pakiet danych (linia *Tx_RAM* w stanie wysokim), sygnał *Pid_End* jest przenoszony przez bramkę G2708 na linię *Buf_Tx_Ena* stanowiąc rozkaz rozpoczęcia pracy dla kolejnych bloków generujących dane i sumę kontrolną. Informacja o zakończeniu generacji przekazywana jest stanem wysokim linii *CRC_End*, umożliwiając bramkom G2703 i G2704 wystawienie sygnału kończ pracy nadajnika *Tx_End*.

7.10 Blok łączenia fragmentów nadawanych pakietów (St_TxMux)

Na rysunku (50) przedstawiono schemat ideowy bloku odpowiedzialnego za połączenie poszczególnych części pakietu danych w jedną całość. Dodatkowym zadaniem realizowanym przez blok jest konwersja danych obieranych z bufora wejściowego magistralą *St_TxData(7:0)* w postaci równoległej na postać szeregową, wymaganą przez Transceiver.



Rysunek 50: Schemat ideowy bloku łączenia fragmentów nadawanego pakietu (**St_TxMux**).

Nadawane dane są podłączone do Transceiver-a poprzez linię *Tx_Data*. W pierwszej fazie pracy multiplekser M2806 przepuszcza na nią nagłówek pakietu, generowany przez **St_Tx** i

przesyłany do bloku linią *Tx_PID*. Przejście linii *Buf_Tx_Ena* w stan wysoki oznacza zakończenie generacji nagłówka i rozpoczęcie nadawania danych z bufora wejściowego bądź, w przypadku trwającego protokołu konfiguracji, z bloku generacji danych informacyjnych **St_DConf**. Dzieje się tak, gdy wystąpi stan wysoki sygnału *Set_Data*, tworzony przez bramkę G2801 ze stanu wysokiego na liniach *Set_Ena* lub *Set_End*. Stan wysoki linii *Set_Data* przełącza multiplekser M2804 na odbiór danych z linii *Tx_DConf*, którą są one przesyłane z bloku **St_DConf**. Z wyjścia multipleksera M2804 dane są przesyłane równolegle linią *Tx_CRCData* na wejście multipleksera M2805 oraz do bloku obliczającego cykliczny kod nadmiarowy **St_TxCRC**. Jeden z sygnałów *TxRAM_End* i *DConf_End*, oznaczających zakończenie pracy przez, odpowiednio, bufor wejściowy **St_Buff** i generator opisów **St_DConf**, jest przekazywany na linię *Buf_Tx_End* przez multiplekser M2807 w zależności od stanu linii *Set_Data*. Do czasu zakończenia pracy jednego z w/w bloków linia *Buf_Tx_End* pozostaje w stanie niskim, dzięki czemu multiplekser M2805 przepuszcza dane z linii *Tx_CRCData* na wejście multipleksera M2806. Ponieważ linia *Buf_Tx_Ena* znajduje się w stanie wysokim, dane są wystawiane przez multiplekser M2806 na linię *Tx_Data* i zostają przesłane do Transceiver-a. Z chwilą zakończenia pracy przez **St_Buff** lub **St_DConf** w stan wysoki przechodzi linia *Buf_Tx_End* nakazując blokowi **St_TxCRC** nadanie obliczonego cyklicznego kodu nadmiarowego. Kod zostaje przesłany linią *Tx_CRC* i przekazany wysokim stanem *Buf_Tx_End* na wyjście multipleksera M2805 skąd, podobnie jak wcześniej dane, jest przesyłany do Transceiver-a.

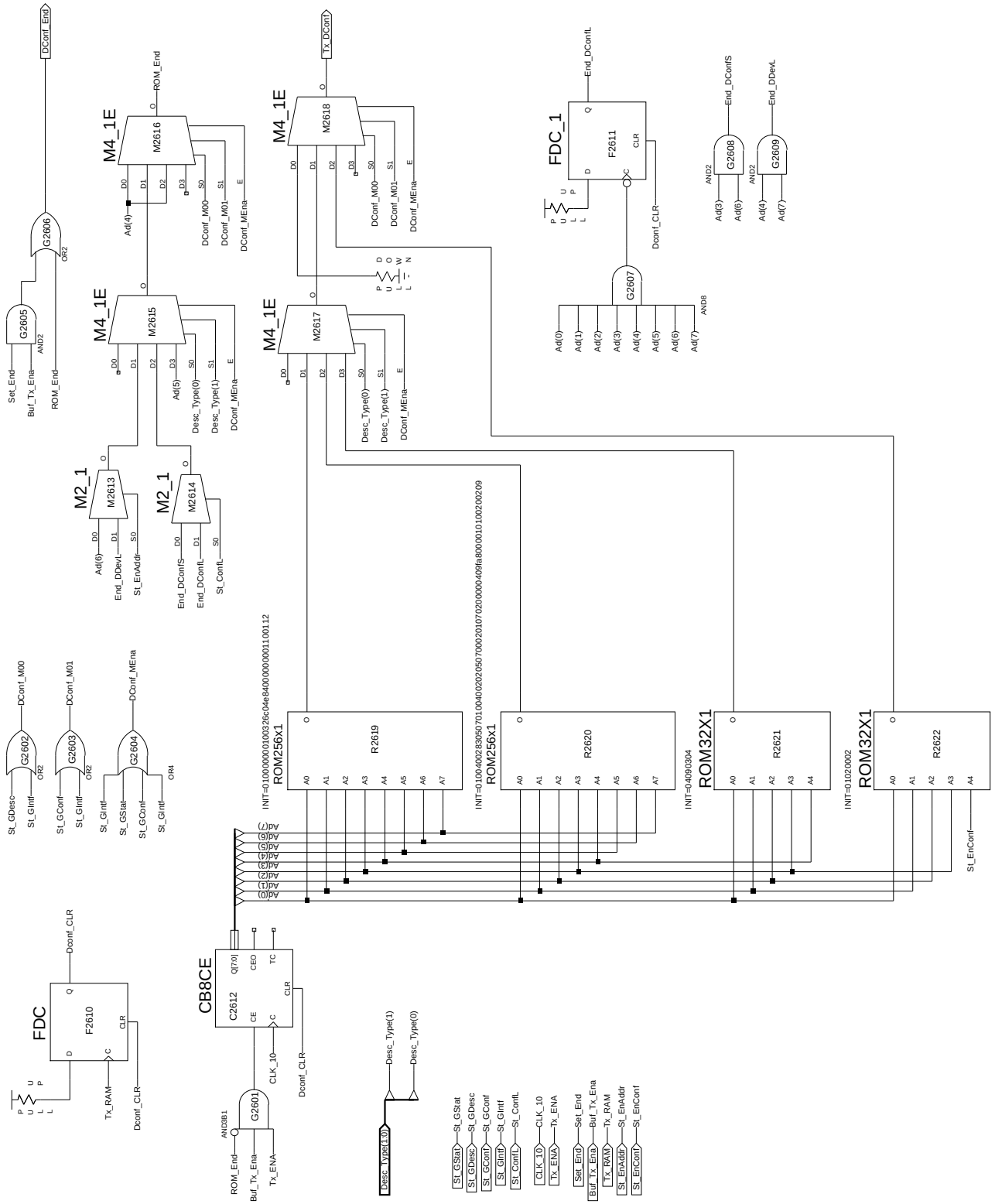
W przypadku odbioru danych z bufora wejściowego **St_Buff**, niski stan na linii *Set_Data* wraz z wysokim stanem sygnałów *Buff_Tx_Ena*, zezwalającego na nadawanie danych, oraz *Tx_ENA*, informującego o gotowości Transceiver-a do ich odbioru, wystawia przez bramkę G2802 sygnał zezwalający na pracę licznika C2803 i multipleksera M2808. Bajt danych przesyłany magistralą *St_TxData(7:0)* z bufora wejściowego **St_Buff** jest podawany na wejścia *D0-D7* multipleksera M2808. Licznik C2803, poprzez wyjścia *Q0-Q3*, wybiera kolejne bity bajtu danych, zwiększając numer bitu na zboczu sygnału zegarowego *CLK_10*. Ze względu na zabieg "skrzyżowania" magistral danych[25] w bloku pamięci **St_RAM64**, LSB bajtu znajduje się na linii *St_TxData(7)* magistrali danych. Zgodnie z kierunkiem transmisji bajtów magistralą USB, bit ten powinien zostać wysłany jako pierwszy, dlatego też linia *St_TxData(7)* została podłączona do wejścia *D0*, wybieranego przez licznik C2803 w pierwszej kolejności. Dane z wyjścia *O* multipleksera M2808 są przesyłane linią *TxSData* poprzez multipleksery M2804, M2805 i M2806, sterowane odpowiednimi poziomami sygnałów, na linię *Tx_Data* skąd są pobierane przez Transceiver.

Ponieważ pierwszy bajt danych jest wystawiany na magistralę *St_TxData(7:0)* od razu po zakończeniu zapisu bloku pamięci bufora wejściowego, zaś LSB tego bajtu jest wybierane stanem początkowym 000B wyjść *Q0-Q3* licznika C2803, bit ten jest gotowy do nadania od razu po wystąpieniu sygnału zezwolenia z bramki G2802. Pobranie wartości bitu przez Transceiver następuje na zboczu zegara *CLK_8*[26], przed wystąpieniem zbocza *CLK_10* zwiększającego wskazanie licznika C2803, co zmienia wybierany przez multiplekser M2808 bit na wyższy. Po osiągnięciu przez licznik C2803 adresu 1000B, stan wysoki wyjścia *Q3* powoduje skasowanie wartości licznika i wybór przez multiplekser M2808 ponownie najmłodszego bitu. Równocześnie wysoki stan wyjścia *Q3* jest przenoszony poprzez linię *CLK_Tx* do bloku bufora wejściowego **St_Buff** powodując zwiększenie adresu pamięci i przesłanie magistralą *St_TxData(7:0)* kolejnego bajtu danych, którego LSB jest przekazywany do Transceiver-a poprzez ustawiony w stan początkowy multiplekser M2808. Osiągnięcie końcowego adresu bloku pamięci bufora wejściowego jest sygnalizowane stanem wysokim linii *TxRAM_End*, co kończy procedurę nadawania danych oraz, po przesłaniu przez blok **Tx_CRC** sumy kontrolnej, całą transmisję nadawczą.

[25]Podrozdział 7.4 niniejszej pracy

[26]Podrozdział 5.5 niniejszej pracy

7.11 Blok generacji danych informacyjnych o interfejsie (St_DConf)



Rysunek 51: Schemat ideowy bloku generacji danych informacyjnych o interfejsie (**St_DConf**).

Rysunek (51) przedstawia schemat ideowy bloku generującego dane informacyjne stanowiące odpowiedzi na żądania[27] przesłane przez komputer w bloku rozkazów protokołu konfiguracji. Odpowiednie opisy interfejsu i jego konfiguracji zawarto w blokach pamięci ROM R2619, R2620 i R2622.

Generacja danych odbywa się poprzez podanie na wejścia adresowe *A0-Ai*, gdzie *i* zależy od ilości słów pamięci, adresu kolejnego bitu odczytywanego z wyjścia *O* danego bloku ROM. Za adresowanie pamięci odpowiada licznik C2612 zwiększający adres na zboczu sygnału zegara *CLK_10* po otrzymaniu zezwolenia ze strony bramki G2601. Wystawia ona sygnał wysoki w czasie, gdy linie zezwalająca na generację danych *Buf_tx_Ena* oraz informująca o gotowości Transceiver-a do ich odbioru *Tx_ENA* znajdują się w stanie wysokim, zaś sygnał zakończenia generacji danych informacyjnych *ROM_End* w niskim. Licznik C2612 jest ustawiany w stan początkowy krótkim impulsem dodatnim *Dconf_CLR*, generowanym przez przerzutnik F2610 na narastającym zboczu sygnału *Tx_RAM* oznaczającym rozpoczęcie tworzenia pakietu danych.

W pamięci ROM R2619 zapisano 18-bajtowy deskryptor typu 'DEVICE', zawierający podstawowy opis interfejsu. Opis znaczenia tego deskryptora zawarto w tabeli 9.

Numer bajtu	Wartość bajtu	Znaczenie
1	12H	Długość deskryptora - 18 bajtów
2	01H	Typ deskrypora - 'DEVICE'
4,3	0110H	Standard specyfikacji USB - 1.10 (Full-Speed)
5	00H	Klasa interfejsu. 0 = określana deskryptorem konfiguracji
6	00H	Podklasa interfejsu. 0 gdy klasa interfejsu ma wartość 0
7	00H	Protokół obsługi interfejsu. 0 jw.
8	40H	Maksymalny rozmiar pakietu danych kontorlnych - 64 bajty
10,9	04E8H	Kod producenta - Samsung Electronics Co., Ltd
12,11	326CH	Wewnętrzny kod produktu producenta (bez znaczenia)
14,13	0100H	Wersja rozwojowa interfejsu - 1.00
15	00H	Indeks deskryptora 'STRING' z nazwą producenta - 0 = brak
16	00H	Indeks deskryptora 'STRING' z nazwą interfejsu - jw.
17	00H	Indeks deskryptora 'STRING' z numerem seryjnym - jw.
18	01H	Ilość możliwych konfiguracji - 1

Tabela 9: Deskryptor podstawowego opisu interfejsu.

Uzyskanie własnego kodu producenta nie było możliwe ze względu na wysokie koszty takiej procedury. Ponieważ wystąpiły trudności z uzyskaniem dostępu do bazy kodów procentów tak, by było możliwe zgłoszenie interfejsu w sposób bardziej odpowiadający rzeczywistości (np. jako firma Xilinx, klasa (określana w deskryptorze konfiguracyjnym) - interfejs wymiany danych), na potrzeby testów przepisano kody producenta i typu interfejsu z deskryptora drukarki jednego z autorów pracy. W końcowej wersji interfejsu, wykonanej poza ramami pracy inżynierskiej, wpisy te zostaną zmienione.

[27]Tabela 6, podrozdział 7.7 niniejszej pracy

W pamięci ROM R2621 zawarto 4-bajtową deklarację użycia języka angielskiego jako języka opisów tekstowych interfejsu na potrzeby komunikatów systemu operacyjnego. Znaczenie poszczególnych bajtów deklaracji przedstawiono w tabeli 10.

Numer bajtu	Wartość bajtu	Znaczenie
1	04H	Długość deklaracji - 4 bajty
2	03H	Typ deskryptora - 'STRING'
4,3	0409H	Język opisów - Angielski(Stany Zjednoczone)

Tabela 10: Deklaracja języka opisów tekstowych interfejsu.

Blok pamięci B2622 zawiera dwie różne, dwubajtowe odpowiedzi na żądanie przedstawienia aktualnie ustawionej konfiguracji interfejsu. Pamięć podzielono na dwa banki wybierane linią *St_EnConf*, podłączoną do najstarszego bitu adresowego pamięci *A4*. Stan niski tej linii, sygnalizujący trwający proces enumeracji wybiera dwa młodsze bajty (pierwszy bank), stan wysoki - dwa starsze (bank drugi). Znaczenie obu odpowiedzi przedstawiono w tabeli 11.

Numer banku	Numer bajtu	Wartość bajtu	Znaczenie
1	1	02H	Długość odpowiedzi - 2 bajty
1	2	00H	0 - nie ustawiono konfiguracji (trwa enumeracja)
2	1	H	Długość odpowiedzi - 2 bajty
2	2	H	Ustawiono konfigurację numer 1

Tabela 11: Informacja o ustawionej konfiguracji interfejsu.

W bloku ROM R2620 zawarto 32-bajtowy deskryptor typu 'CONFIGURATION' zawierający pełny opis konfiguracji interfejsu i jego składowych. Opis znaczenia tego deskryptora przedstawiono w tabeli 12.

Odpowiedzią interfejsu na żądanie przedstawienia statusu są dwa bajty zawierające same zera[28]. Z tego względu nie umieszczono piątego bloku ROM, lecz podłączono odpowiadające temu żądaniu wejście *D0* multipleksera M2618 na stałe do masy.

Podczas pracy licznika C2612 adres jest podawany równolegle do wszystkich bloków ROM, przez co wystawiają one równocześnie zawartość odpowiednich komórek pamięci na wyjścia *O*. Wybór odpowiedniego wyjścia, a, co za tym idzie, rodzaju generowanej odpowiedzi, umożliwiają multipleksery M2617 i M2618. Sygnał zezwalający na ich pracę *DConf_MEna*, tworzony jest przez bramkę G2604 na podstawie stanu wysokiego jednej z linii przekazujących rozpoznane przez **St_Brqst** żądzanie. Bramki G2602 i G2063 tworzą koder "4 do 2" pracujący zgodnie z tablicą prawdy przedstawioną tabeli 13.

[28]Universal Serial Bus Specification, rev. 2.0, rozdz. 9.4.5, s. 254

Numer bajtu	Wartość bajtu	Znaczenie
Podstawowy opis interfejsu		
1	09H	Długość pierwszej części opisu - 9 bajtów
2	02H	Typ deskryptora - 'CONFIGURATION'
4,3	0020H	Całkowita długość opisu - 32 bajty
5	01H	Ilość interfejsów - 1
6	01H	Ilość możliwych konfiguracji - 1
7	00H	Indeks deskryptora 'STRING' z opisem konfiguracji - 0 = brak
8	80H	Interfejs zasilany z magistrali USB
9	FAH	Maksymalny pobór prądu - 500mA
Opis konfiguracji interfejsu		
1	09H	Długość opisu konfiguracji interfejsu - 9 bajtów
2	04H	Typ deskryptora - opis kofiguracji interfejsu
3	00H	Numer interfejsu - 0
4	00H	Ilość alternatywnych konfiguracji - 0
5	02H	Ilość buforów - 2
6	07H	Klasa interfejsu - urządzenie drukujące
7	01H	Podklasa interfejsu - drukarka
8	02H	Protokół wymiany danych - dwukierunkowy
9	00H	Indeks deskryptora 'STRING' z opisem interfejsu - 0 = brak
Opis konfiguracji bufora wejściowego danych		
1	07H	Długość opisu konfiguracji bufora wejściowego- 7 bajtów
2	05H	Typ deskryptora - opis konfiguracji bufora
3	02H	Numer i typ bufora - 2, nadawanie danych do komputera
4	02H	Typ protokołu transmisji - 'Bulk'
6,5	0040H	Rozmiar bufora - 64 bajty
7	01H	Możliwość odrzucania pakietów danych - tak
Opis konfiguracji bufora wyjściowego danych		
1	07H	Długość opisu konfiguracji bufora wyjściowego- 7 bajtów
2	05H	Typ deskryptora - opis konfiguracji bufora
3	83H	Numer i typ bufora - 3, odbiór danych z komputera
4	02H	Typ protokołu transmisji - 'Bulk'
6,5	0040H	Rozmiar bufora - 64 bajty
7	01H	Możliwość odrzucania pakietów danych - tak

Tabela 12: Deskryptor pełnego opisu konfiguracji interfejsu.

-	Wejścia bramek			Sygnały wyjściowe	
St_GStat	*St_GDesc*	*St_GConf*	*St_GIntf*	*DConf_M01*	*DConf_M00*
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

Tabela 13: Tablica prawdy kodera "4 do 2" (G2602 i G2603).

Sygnały *DConf_M00* i *DConf_M01*, doprowadzone do wejść wybierających multipleksera M2618 umożliwiają mu przepuszczenie odpowiedzi, kolejno, o statusie interfejsu (stały stan niski), opisu interfejsu (o typie wybieranym przez multiplekser M2617), oraz o stanie konfiguracji interfejsu (z pamięci R2622). Na podstawie kombinacji stanów linii magistrali *Desc_Type(1:0)*[29] multiplekser M2617 wybiera odpowiedni rodzaj deskryptora - typu 'DEVICE' z pamięci R2519, 'CONFIGURATION' z R2620 oraz 'STRING' z R2621. Wybrane w wyżej opisany sposób dane są przekazywane z wyjścia multipleksera M2618 do bloku **St_TxMux** linią *Tx_DConf*.

Ponieważ każdy z opisów ma inną długość, podobną konstrukcję multiplekserów musiano zastosować w celu uzyskania syganłu *ROM_End* informującego o zakończeniu generacji danych informacyjnych, czyli o odczycie całości zapisanych w danej pamięci ROM danych. Multiplekser M2616, odpowiadający M2618, wystawia stan wysoki na wyjściu po osiągnięciu przez licznik C2612 wartości będącej adresem o jeden wyższym od adresu ostatniego bitu danego deskryptora. Dla dwubajtowych (czyli szesnastobitowych) deskryptorów statusu (wejście *D0* multipleksera M2616) i ustawionej konfiguracji (*D2*) sygnałem tym będzie stan wysoki na piątym bicie *Ad(4)* magistrali licznika (10000B = 16). Długość opisu interfejsu, na podstawie jego typu, wybiera multiplekser M2615 (analog M2617). Dla deklaracji języka ('STRING', wejście *D3*) długości 4 bajtów (32 bitów) koniec zapisu sygnalizuje stan wysoki szóstego bitu *Ad(5)* magistrali licznika (100000B = 32). Ponieważ opis konfiguracji (wejście *D2* multipleksera M2615) ma zmienną długość, zależną od wartości zadeklarowanej w rozkazie[30], a przesyłanej do bloku linią *St_ConfL*, sygnał zakończenia jest wybierany przez multiplekser M2614 z linii *End_DConfS* (dla deskryptora krótkiego, 9-bajtowego) i *End_DConfL* (dla długiego, 32-bajtowego). Sygnał *End_DConfS* jest tworzony przez bramkę G2608 na postawie wysokiego stanu czwartego (*Ad(3)*) i siódmego (*Ad(6)*) bitu magistrali licznika C2612, co odpowiada liczbie 1001000B = 72, czyli ośmiu bitom. Ponieważ sygnał *End_DConfL* powinien odpowiadać liczbie $32 * 8$=256, czyli 100000000B, wymagałoby to pobrania stanu wysokiego z nieistniejącego, dziewiątego bitu licznika. Zamiast tego zastosowano przerzutnik F2611 wystawiający stan wysoki na linię *End_DConfL* w chwili przejścia magistrali licznika C2612 ze stanu 11111111B=255 (stan wysoki na wyjściu bramki G2607) do 00000000B (stan niski). Stan przerzutnika F2612 kasowany jest sygnałem *Dconf_CLR*. Podobna sytuacja, o czym wcześniej nie wspomniano, występuje dla deskryptora typu 'DEVICE', zawierającego podstawowy opis interfejsu. Pełne 18 bajtów wysyłane jest dopiero w stanie zaadresowania interfejsu (linia *St_EnAddr* w stanie wysokim). Wcześniej, przed przypisaniem adresu przez komputer, wysyłane powinno być jedynie pierwsze osiem bajtów deskryptora. Zapewnia to multiplekser M2613 podający na wejście *D2* multipleksera M2615 odpowiadające temu typowi opisu właściwy sygnał końca odczytu. Dla krótkiego deskryptora (wejście *D0* M2613) jest to siódmy bit *ad(6)* magistrali licznika C2612, odpowiadający adresowi 1000000B=64 (czyli ośmiu bajtom). Sygnał końca 18-bajtowego deskryptora,

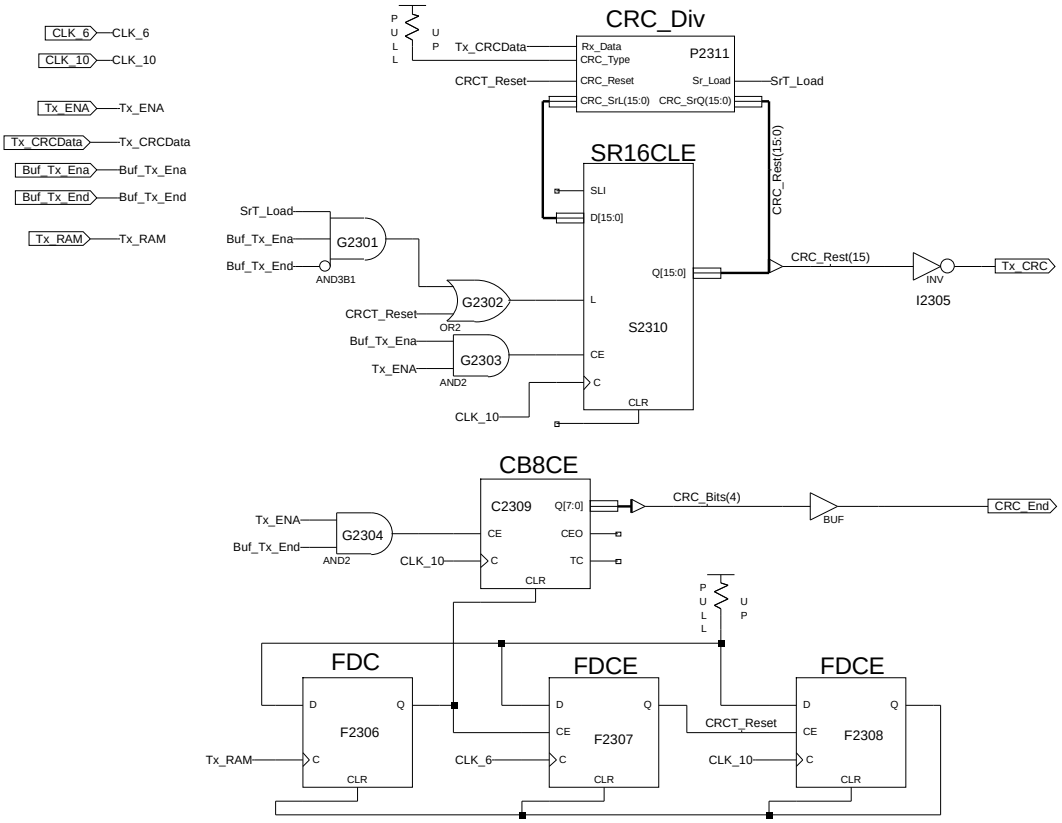
[29]Tabela 7, podrozdział 7.7 niniejszej pracy

[30]Podrozdział 7.7 niniejszej pracy

End_DDevL jest tworzony przez bramkę G2609 przy wysokim stanie piątego (*Ad(4)*) i ósmego (*Ad(7)*) bitu magistrali licznika C2612, co odpowiada liczbie 10010000B=144, czyli 18 bajtom.

Sygnał zakończenia generacji danych *ROM_End* jest przenoszony przez bramkę G2606 na linię *DConf_End*, którą trafia do bloku łączącego części pakietu **St_TxMux**. Drugą możliwością ustawienia sygnału *DConf_End* jest generacja pakietu 'Data zero length' przy procedurze kończącej protokół konfiguracji. Wówczas linia *Set_End* znajduje się w stanie wysokim, co natychmiast po wystawieniu logicznej jedynki na linii *Buf_Tx_Ena* jest przenoszone przez bramki G2605 i G2606 na linię *DConf_End*. W ten sposób omijana jest generacja danych i utworzony zostaje pakiet o zerowej ich długości, czyli 'Data zero length'

7.12 Blok obliczania sumy kontrolnej CRC pakietów nadawanych (St_TxCRC)



Rysunek 52: Schemat ideowy bloku obliczania CRC pakietów nadawanych (**St_TxCRC**).

Na rysunku (52) przedstawiono schemat bloku obliczającego cykliczny kod nadmiarowy dla danych wychodzących. Algorytm obliczania tego kodu jest taki sam, jak w przypadku bloku **St_RxCRC**[31]. Na narastającym zboczu sygnału rozpoczęcia transmisji *Tx_RAM* przerzutnik F2306 ustawia stan wysoki na wyjściu *Q* kasując zawartość licznika C2309 i umożliwiając przerzutnikowi F2307 wystawienie stanu wysokiego na linii *CRCT_Reset* na zboczu sygnału zegarowego *CLK_6*. Stan wysoki linii *CRCT_Reset* ustawia przez bramkę G2302 zezwolenie dla rejestru przesuwnego S2310 na przepisanie zawartości magistrali wyjściowej *CRC_SrL(15:0)* bloku **CRC_Div** na zboczu sygnału zegarowego *CLK_10*. Magistrala ta, ze względu na stan linii *CRCT_Reset*, zawiera same jedynki na wszystkich bitach, ładując wstępnie rejestr S2310.

[31] Podrozdział 7.2 niniejszej pracy

Równocześnie przerzutnik F2308 na tym samym zboczu *CLK_10* generuje krótki impuls kasujący przerzutniki F2306 i F2307.

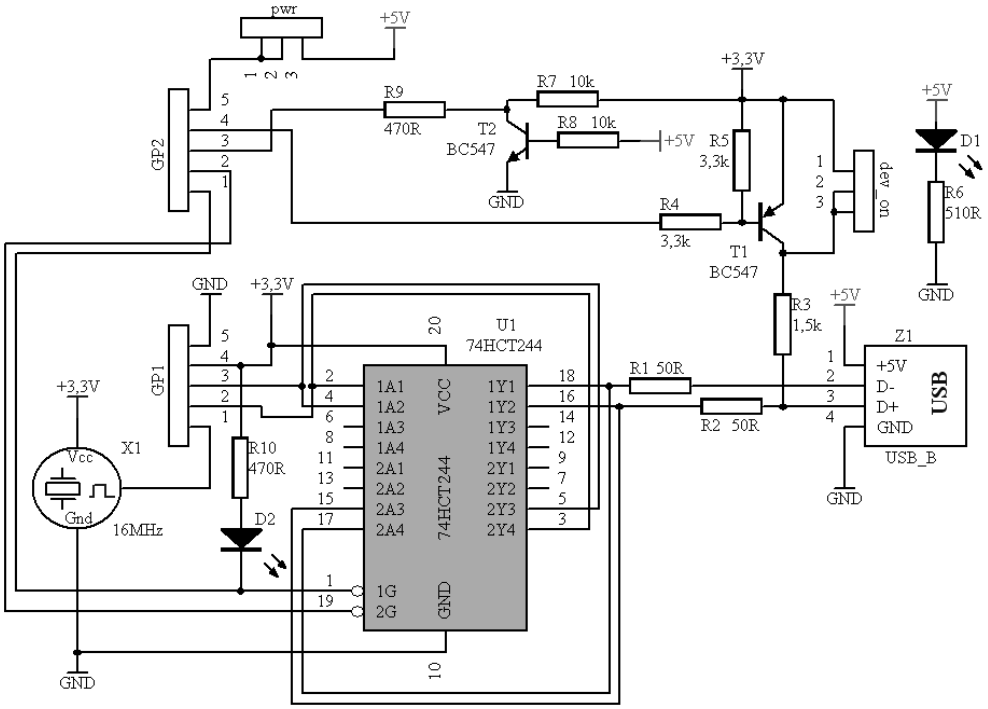
Bramka G2303 wystawia sygnał zezwolenia na pracę rejestru S2310 wtedy, gdy generowane są dane stanowiące treść pakietu (linie *Buf_Tx_Ena* i *Tx_ENA* w stanie wysokim). Dane te, poprzez linię *Tx_CRCData* są przekazywane do bloku **CRC_Div**, gdzie są porównywane z najstarszym bitem rejestru. Na podstawie porównania podejmowana jest decyzja o załadowaniu (sygnał *SrT_Load* w stanie wysokim) rejestru S2310 wynikiem dzielenia jego zawartości przez wielomian generacyjny (dokładny opis tej procedury znajduje się w podrozdziale 7.2 niniejszej pracy). Ponieważ interfejs nie generuje pakietów typu 'Token', wejście *CRC_Type* bloku **CRC_Div** jest na stałe podpięte do stanu wysokiego, co oznacza obliczanie szesnastobitowego kodu CRC16 dla pakietów typu 'Data'. Bramka G2301 blokuje sygnał ładowania *SrT_Load* w sytuacji, gdy generowany jest nagłówek pakietu (linia *Buf_Tx_Ena* w stanie niskim) lub gdy zakończyło się nadawanie danych (linia *Buf_Tx_End* w stanie wysokim).

Otrzymanie sygnału zakończenia generacji danych przez bufor wejściowy **St_Buff** lub generator opisów **St_DConf** oznacza, że pozostały w rejestrze S2310 wynik obliczeń cyklicznego kodu nadmiarowego powinien zostać, po zanegowaniu, przesłany linią *Tx_CRC* w celu dołączenia na koniec tworzonego pakietu. Ponieważ na linię *Tx_CRC* wystawiany jest poprzez inwerter I2305 najstarszy bit *CRC_Rest(15)* magistrali wyjściowej rejestru S2310, pierwszy bit kodu, pobierany przez Transceiver na zboczu sygnału zegarowego *CLK_8*[32] jest gotowy do transmisji natychmiast po zakończeniu generacji danych przez poprzednie bloki (wyjątkowo cykliczny kod nadmiarowy jest transmitowany w kolejności MSB → LSB). Zbocze zegara *CLK_10*, nadchodzące tuż po *CLK_8*, przesuwa zawartość rejestru S2310 o jedną komórkę w przód, wystawiając tym samym negację kolejnego bitu cyklicznego kodu nadmiarowego na linię *Tx_CRC*. Za przesłanie dokładnie szesnastu bitów kodu odpowiada licznik C2309 zliczający takty zegara *CLK_10* przesuwające kod nadmiarowy w komórkach rejestru. Sygnał zezwolenia jest tworzony bramką G2304 z sygnału zakończenia transmisji danych *Buf_Tx_End* i sygnału informującego o możliwości pobierania danych przez Transceiver *Tx_ENA*. Wystawienie stanu wysokiego na piątym bicie *CRC_Bits(4)* magistrali wyjściowej licznika, odpowiadające wartości 10000B=16, oznacza szesnastokrotne przesunięcie zawartości rejestru S2310 i wystawienie wszystkich bitów cyklicznego kodu nadmiarowego na linię *Tx_CRC*. Sygnał ten jest przesyłany linią *CRC_End* do bloku **St_Tx**, który na jego podstawie kończy generację i nadawanie kompletnego pakietu danych.

[32]Podrozdział 5.5 niniejszej pracy

8 Zewnętrzny układ wykonawczy

Celem pracy było zbudowanie takiego interfejsu USB, aby był w całości wykonany wewnątrz struktury układu programowalnego (FPGA) i zawierał tylko bezwzględnie konieczne do działania elementy zewnętrzne. Po wielu testach powstał dodatkowy układ zewnętrzny wyposażony w najpotrzebniejsze podzespoły. Widok tego dodatkowego modułu przedstawiono na rysunku (53). Głównym jego zadaniem jest dopasowywanie parametrów sygnału przy transmisji między komputerem a układem programowalnym, możliwość odłączenia płytki od komputera (bez wyjmowania kabla USB) i wykrycie czy kabel USB jest wpięty do komputera. Po późniejszych testach moduł wykonawczy został wyposażony w stabilny generator kwarcowy, dostarczający przebiegu prostokątnego o częstotliwości $16MHz \pm 0,25\%$ oraz diody sygnalizujące stan pracy nadajnika i obecność zasilania na porcie USB (włączenie wtyczki do portu).



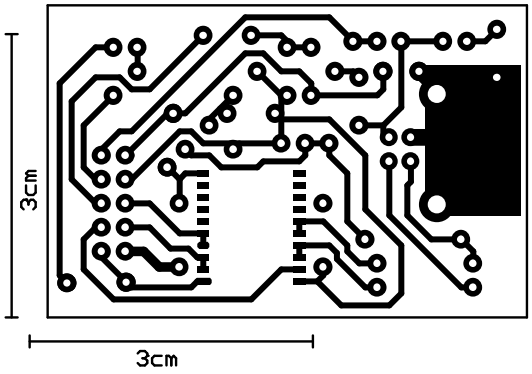
Rysunek 53: Schemat ideowy zewnętrznego układu wykonawczego

Złącza GP1 i GP2 tworzą razem gniazdo o wymiarach 2x5 styków zapewniające połączenie układu wykonawczego z płytką na której znajduje się programowalny układ logiczny (FPGA). Za pomocą tego złącza cała płytka może być zasilana z komputera (dzięki zworce pwr). Sygnał z buforów wyjściowych układu FPGA trafia na układ U1 (74HC244). Jest to ośmiobitowy bufor trójstanowy (8 pojedynczych buforów w jednej obudowie), podłączony w sposób umożliwiający obustronną propagację sygnałów. Układ scalony U1 zawiera dwie grupy buforów, z których każda ma osobny sygnał zezwalający (sygnały 1G i 2G). Dzięki temu połączono wejścia jednej grupy buforów z wyjściami drugiej grupy i uzyskano dwa kanały do dwukierunkowego przesyłania danych. Jako sterowanie kierunkiem przepływu wykorzystano komplementarny sygnał na liniach 1G i 2G ($1G = \overline{2G}$), generowany za pomocą układu FPGA. Użycie dodatkowego układu buforującego (U1) jest konieczne ze względu na parametry sygnału, jakie muszą być zapewnione. Sam układ programowalny generuje zbyt szybkie zbocza sygnału, które nie mieszczą się w założeniach przewidzianych w specyfikacji USB, natomiast układ 74HC244 posiada gwarantowane parametry sygnału trafiające dokładnie w środek stawianych wymagań.

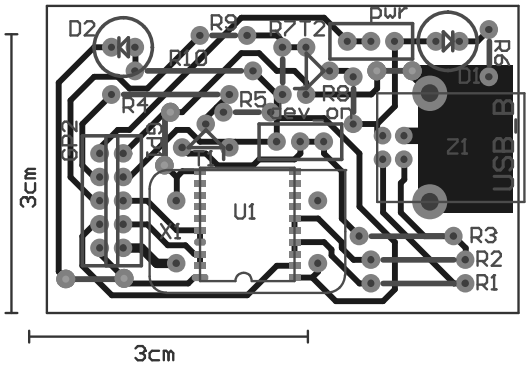
Rezystory R1 i R2 zapewniają prawidłowe warunki pracy na magistrali USB, ustalając wartość prądu mieszczącą się w koniecznych wymaganiach magistrali. Sygnał danych wyprowadzony jest na gniazdo typu B, będące jednym ze standardów wśród urządzeń pracujących na USB. Aby opisywane urządzenie mogło zostać poprawnie rozpoznane na magistrali jako nowe urządzenie pracujące w trybie Full-Speed, wymagane jest dołączenie rezystora podciągającego linię danych D+ do napięcia +3.3V. Rolę tą pełni rezystor R3, jednak został on dołączony do napięcia zasilającego za pomocą obwodu z tranzystorem T1 i dodatkowymi rezystorami R4 i R5. Taki sposób podłączenia umożliwia odpięcie urządzenia od magistrali USB w dowolnym momencie za pomocą stanu logicznego na bazie tranzystora T1. Gdy układ FPGA wystawi za pośrednictwem rezystora R4 stan niski na bazie T1, to urządzenie zostanie podłączone, natomiast ustawienie stanu wysokiego odłączy urządzenie. Gdyby automatyczne odłączanie urządzenia stało się niepotrzebne, można na stałe dołączyć rezystor R3 za pomocą zworki dev_on.

Tranzystor T2 oraz rezystory R7, R8 i R9 tworzą prosty obwód umożliwiający wykrycie, czy do modułu wykonawczego jest podłączone napięcie za pośrednictwem kabla USB (czy moduł jest włączony do komputera). Ma to znaczenie, gdy płytka z układem FPGA nie jest zasilana za pośrednictwem modułu wykonawczego bezpośrednio z komputera, a posiada swój własny zasilacz. O podłączeniu kabla USB do komputera świadczy stan niski na kolektorze tranzystora T2.

Płytka drukowana została zaprojektowana jako jednowarstwowa ze względu na prostotę układu. Ścieżki przedstawiono na rysunku (54) a schemat montażowy na rysunku (55).



Rysunek 54: Rysunek płytki układu wykonawczego



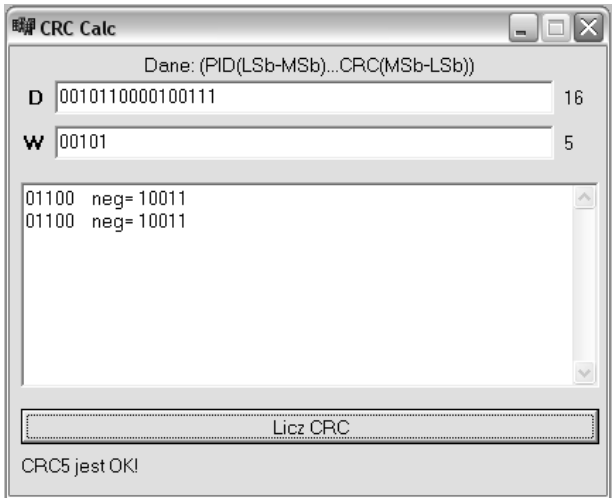
Rysunek 55: Schemat montażowy układu wykonawczego

9 Testowanie i analiza działania interfejsu USB

Zaprojektowanie interfejsu komunikacyjnego nie byłoby możliwe bez dodatkowych programów i urządzeń, które stworzono w czasie pracy nad właściwym projektem. Na szczególną uwagę zasługują program do obliczania cyklicznego kodu nadmiarowego (CRC) i analizator stanów logicznych zbudowany w oparciu o mikrokontroler ATMEGA162[33].

9.1 Program do obliczania kodu CRC (CRCCalc)

Program CRCCalc napisano w darmowej wersji $C + Builder6$[34]. Jego zadaniem jest obliczanie cyklicznego kodu nadmiarowego[35] (CRC) z danych (D) na podstawie podanego wielomianu generacyjnego (W), lub sprawdzanie poprawności pakietu danych pod względem zgodności CRC. Na rysunku (56) przedstawiono wygląd programu.



Rysunek 56: Wygląd programu liczącego CRC

Program był szczególnie przydatny w początkowej fazie projektowania, gdy bloki logiczne liczące i analizujące kod CRC nie były jeszcze gotowe, a konieczna była analiza poprawności otrzymywanych danych.

9.2 24-bitowy analizator stanów logicznych

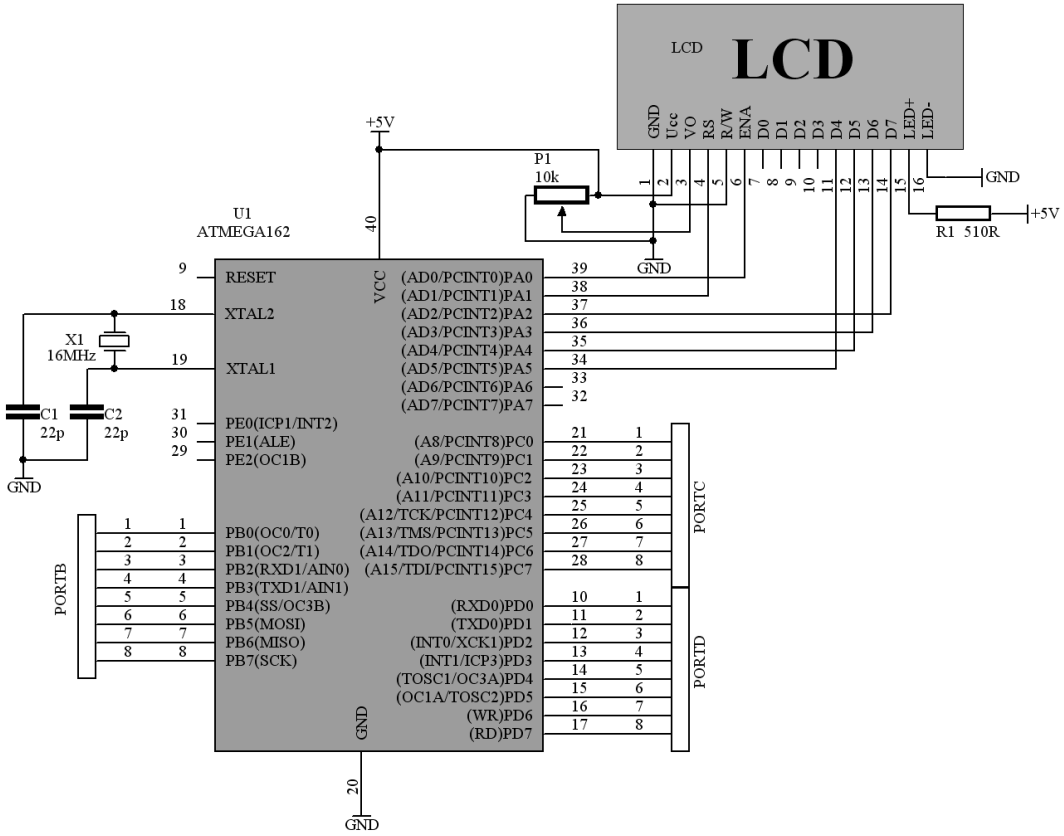
Bardzo przydatnym urządzeniem, wykorzystywanym przez cały czas projektowania interfejsu USB, jest 24-bitowy analizator stanów logicznych. Zbudowano go w oparciu o mikrokontroler jednoukładowy ATMEGA162. Schemat układu analizatora przedstawiono na rysunku (57). Główną jego częścią jest wspomniany mikrokontroler, ale analizator posiada również wyświetlacz ciekłokrystaliczny (LCD), na którym prezentowane są odczytywane stany logiczne. Do odczytu poziomów logicznych wykorzystano trzy ośmiobitowe porty mikrokontrolera (24 bity), podłączone bezpośrednio do płytki testowej z układem FPGA.

[33] http://www.atmel.com/dyn/resources/prod_documents/doc2513.pdf, karta katalogowa mikrokontrolera
[34] http://www.borland.com/ - strona firmy Borland
[35] Universal Serial Bus Specification, rev. 2.0, s. 198 - algorytm liczenia CRC

Mikrokontroler jest układem programowalnym, podobnie jak układy FPGA, lecz jego działanie opiera się na wykonywaniu kolejnych rozkazów zawartych w programie, a nie syntezie układu logicznego. Oprogramowanie dla mikrokontrolera napisano w programie BASCOM AVR firmy MCS Electronics[36], a jego kod źródłowy (uproszczony) wygląda następująco:

```
Dim X(3) As Byte,  N As Byte
Do
 If X(1) <> Pinb Or X(2) <> Pinc Or X(3) <> Pind Then
   X(1) = Pinb; X(2) = Pinc; X(3) = Pind
   Cls
   For N = 1 To 3
     Lcd X(n).7 ; X(n).6 ; X(n).5 ; X(n).4 ; X(n).3 ; X(n).2 ; X(n).1 ; X(n).0
     If N = 2 Then Lowerline
   Next N
 End If
Loop
```

Działanie programu polega na wypisywaniu stanów logicznych z portów mikrokontrolera na wyświetlacz LCD. Wypisywanie następuje po wykryciu zmiany stanu dowolnego bitu.



Rysunek 57: Schemat analizatora stanów logicznych

[36]http://www.mcselec.com/ - strona internetowa producenta programu BASCOM AVR

9.3 Testowanie komunikacji z komputerem

Podczas projektowania interfejsu płytka testowa była podłączona do komputera z systemem operacyjnym *ArchLinux*, gdzie za pomocą poleceń *dmesg* i *lsusb* odczytywano konfigurację i stan pracy urządzenia. Pierwsze polecenie zwraca informację w postaci:

```
usb 2-2: new full speed USB device using uhci\_hcd and address 2
usb 2-2: config index 0 descriptor too short (expected 32, got 9)
usb 2-2: config 1 has 0 interfaces, different from the descriptor's value: 1
usb 2-2: configuration #1 chosen from 1 choice
```

Projektowany interfejs zgłasza się zatem jako nowe urządzenie pracujące w standardzie Full-Speed. Zostaje mu nadany adres równy 2. Ponieważ opis (Descriptor) urządzenia nie jest w pełni akceptowany przez komputer, występuje problem z konfiguracją interfejsu. Na samym końcu komputer wybiera konfigurację interfejsu, spośród jednej dostępnej w projektowanym interfejsie.

Polecenie *lsusb* listuje wszystkie urządzenia podłączone do komputera (Hosta) dając wynik w postaci:

```
Bus 001 Device 001: ID 1d6b:0001
Bus 001 Device 024: ID 04e8:326c Samsung Electronics Co., Ltd
Bus 002 Device 001: ID 1d6b:0001
Bus 003 Device 001: ID 1d6b:0001
Bus 004 Device 001: ID 1d6b:0001
Bus 005 Device 001: ID 1d6b:0002
```

Urządzenie o numerze Vendor równym 04e8:326c jest projektowanym interfejsem USB. Po wydaniu polecenia *lsusb* z przełącznikiem *-vd 04e8:326c* uzyskano szczegółowe informacje o urządzeniu w postaci:

```
Bus 001 Device 024: ID 04e8:326c Samsung Electronics Co., Ltd
Device Descriptor:
 bLength                18
 bDescriptorType         1
 bcdUSB               1.10
 bDeviceClass            0 (Defined at Interface level)
 bDeviceSubClass         0
 bDeviceProtocol         0
 bMaxPacketSize0        64
 idVendor           0x04e8 Samsung Electronics Co., Ltd
 idProduct          0x326c
 bcdDevice            1.00
 iManufacturer           0
```

```
iProduct                0
iSerial                 0
bNumConfigurations      1
Configuration Descriptor:
  bLength                 9
  bDescriptorType         2
  wTotalLength           32
  bNumInterfaces          1
  bConfigurationValue     1
  iConfiguration          0
  bmAttributes         0x80
    Bus Powered
  MaxPower            500mA
```

Uzyskano wynik pokrywający się dokładnie z wartościami zadeklarowanymi w pamięci stałej programowanego interfejsu USB. Dowodzi to poprawności działania urządzenia w zakresie stopnia jego aktualnego zaawansowania.

10 Podsumowanie

W ramach pracy przestudiowano specyfikację USB, która ze względu na duży stopień skomplikowania i zawiłość, nie jest łatwą lekturą. Stanowi ona jedynie zbiór zasad i reguł jakie muszą być spełnione, aby urządzenie mogło działać na magistrali USB. Wszystkie opisywane w pracy bloki funkcjonalne nie pochodzą ze specyfikacji, a są wynikiem przemyśleń autorów, w celu podziału projektu na prostsze do zrozumienia części, wykonujące określone zadania. Daje to możliwość łatwiejszej kontroli nad projektem a jednocześnie umożliwia przedstawienie jego działania w zwięzły sposób.

Na potrzeby pracy zapoznano się z obsługą programu ISE 10.1 firmy Xilinx, środowiskiem programistycznym, przeznaczonym między innymi do programowania i konfiguracji programowalnych układów logicznych. Zapoznano się z podstawowymi elementami układów FPGA i nauczono się je wykorzystywać na potrzeby projektu.

Podczas projektowania interfejsu konieczne było jego testowanie na różnych stopniach zaawansowania. Zrobiono w tym celu szereg dodatkowych testowych układów elektronicznych w oparciu o mikrokontrolery jednoukładowe. Były to przede wszystkim generatory i analizatory stanów logicznych. Pierwsze z nich używane były głównie przy budowie Transceiver'a, a drugie przy testowaniu całego interfejsu USB.

Na potrzeby pracy zaprojektowano i wykonano obwody drukowane, zarówno do wspomnianych wcześniej układów testowych, jak i zewnętrznego modułu wykonawczego, wchodzącego w skład pracy. Tylko ostatnią wersję obwodu drukowanego przedstawiono w rozdziale 8.

W czasie projektowania wykorzystano także umiejętności programowania w wielu językach. Do programowania mikrokontrolerów wykorzystano język BASCOM AVR, natomiast do prostych programów obliczeniowych zastosowano C++ Builder 6.

W ramach projektu nawiązano obustronną komunikację z komputerem (Hostem). Interfejs poprawnie interpretuje procedury nadania adresu i przekazuje do komputera niezbędne informacje o swojej konfiguracji. Nie obsługuje jednak jeszcze wszystkich żądań wysyłanych z komputera, więc proces enumeracji nie może się poprawnie zakończyć. Sprawę sterownika (czysto programistyczną) dla systemu operacyjnego odłożono na dalszy plan. Podobnie na dalszy plan została odłożona sprawa stworzenia opisu urządzenia (Device Descriptor[37]) i jego konfiguracji (Configuration Descriptor[38]). Wartości te mają być zapisane w pamięci urządzenia i wysyłane do komputera na każde jego żądanie. Na chwilę obecną opis i konfiguracja urządzenia zostały skopiowane z urządzenia fabrycznego (drukarka firmy Samsung), więc projektowany interfejs jest rozpoznawany w systemie jako drukarka.

Ze względu na stopień skomplikowania projektu, a także wiele dodatkowych, koniecznych do wykonania układów i programów, udało się do tej pory zrealizować pierwszą część interfejsu pracującego na magistrali USB. Dotychczasowy wkład pracy (mierzony także jej objętością) już na obecnym etapie znacznie wykracza poza typową objętość pracy inżynierskiej. Prace nad projektem będą trwały w dalszym stopniu, ale w ramach prywatnych zainteresowań i nie wejdą w zakres niniejszej pracy inżynierskiej.

[37] Universal Serial Bus Specification, rev. 2.0, s. 261

[38] Universal Serial Bus Specification, rev. 2.0, s. 264

11 Literatura

[1-9],[12-21],[23],[28],[35],[37-38] Universal Serial Bus Specification, rev. 2.0 - Specyfikacja USB 2.0

[10] http://direct.xilinx.com/direct/ise10_tutorials/ise10tut.pdf - opis programu ISE 10.1.

[11] http://em.avnet.com/spartan3a-evl - strona internetowa firmy Avnet z informacjami na temat płytki testowej.

[33] http://www.atmel.com/dyn/resources/prod_documents/doc2513.pdf - karta katalogowa mikrokontrolera ATMEGA162

[36] http://www.mcselec.com/ - strona internetowa producenta programu BASCOM AVR

[34] http://www.borland.com/ - strona internetowa firmy Borland