POLITECHNIKA POZNAŃSKA
WYDZIAŁ ELEKTRONIKI I TELEKOMUNIKACJI

Maciej Trawka

TESTOWANIE SYSTEMÓW JEDNOUKŁADOWYCH PRZY WYKORZYSTANIU SZEREGOWEJ TRANSMISJI DANYCH

PRACA MAGISTERSKA

Promotor
dr inż. Artur Pogiel

Poznań, 2013

Streszczenie

W pracy zaproponowano protokół szeregowej transmisji danych testowych, wykorzystujący szybkie łącze SerDes. Protokół ten pozwala na przeprowadzenie testu produkcyjnego układów scalonych wielkiej skali integracji o małej liczbie wyprowadzeń, które podczas normalnej pracy wykorzystują wspomniane łącza SerDes do komunikacji z innymi układami wchodzącymi w skład większego systemu. W pracy podjęto próbę pokazania funkcjonowania zaproponowanego protokołu w różnych konfiguracjach, między innymi testu wielu układów jednocześnie, czy pracy z dynamiczną alokacją kanałów testowych. Poprawność działania protokołu we wszystkich konfiguracjach została sprawdzona praktycznie poprzez implementację w układach FPGA.

Abstract

This thesis proposes a protocol of serialized test data transmission that employs a high speed SerDes interface. The presented approach allows the manufacturing test of Very-Large Scale of Integration (VLSI) circuits featuring low pin count at the same time implementing a SerDes interface for the functional use. In the thesis, several configurations of the serialized test data transmission have been presented that include testing of System-on-Chip or with the use of dynamic allocation of the tester channels. All the proposed schemes have been validated using Field-Programmable Gate Arrays.

Spis treści

Spis użytych akronimów

ATE – ang. *Automatic Test Equipment*

At-Speed – metoda przeprowadzania testu układu scalonego przy nominalnej, zgodnie z parametrami jego pracy, częstotliwości sygnału taktującego

BER – ang. *Bit Error Rate*

CRC – ang. *Cyclic Redundancy Check*

DFT – ang. *Design For Testability*

DUT – ang. *Device Under Test*

EDT – ang. *Embedded Deterministic Test*

FPGA – ang. *Field-Programmable Gate Array*

JTAG – ang. *Joint Test Access Group*

LPCT – ang. *Low-Pin Count Test*

LUT – ang. *Look-Up Table*

OCC – ang. *On-chip Clock Controller*

SerDes – ang. *Serializer – Deserializer*

SoC – ang. *System-on-Chip*

STIL – ang. *Standard Test Interface Language*

TAP – ang. *Test Access Port*

VLSI – ang. *Very Large Scale of Integration*

WGL – ang. *Waveform Generation Language*

1. Wstęp

1.1. Wprowadzenie

Minęło wiele lat od czasu wynalezienia pierwszego tranzystora. Rozwój układów elektronicznych, a szczególnie rosnąca złożoność struktur scalonych sprawia, że coraz trudniej je testować. Dużym wyzwaniem dla inżynierów jest takie zaprojektowanie układu scalonego, by można było na etapie produkcji z jak największą precyzją sprawdzić, czy układ nie zawiera uszkodzeń, a jednocześnie, by układ wzbogacony o dodatkowe struktury służące wyłącznie testowaniu nie cechował się gorszymi od założonych parametrami.

W czasach dominującej gospodarki rynkowej jednym z najważniejszych parametrów wpływających na rozwój firm jest koszt produkcji [1]. Dla producentów scalonych układów elektronicznych bardzo ważne jest zatem, by test produkcyjny każdego wytworzonego egzemplarza był tak dokładny, jak to tylko możliwe i przeprowadzony w możliwie krótkim czasie. Dokładność jest ważna z uwagi na silną, na rynku elektroniki, konkurencję, a czas testu przekłada się bezpośrednio na koszt produkcji – im mniej czasu trwa procedura testowa, tym jest mniej kosztowna, zgodnie z porzekadłem „czas to pieniądz".

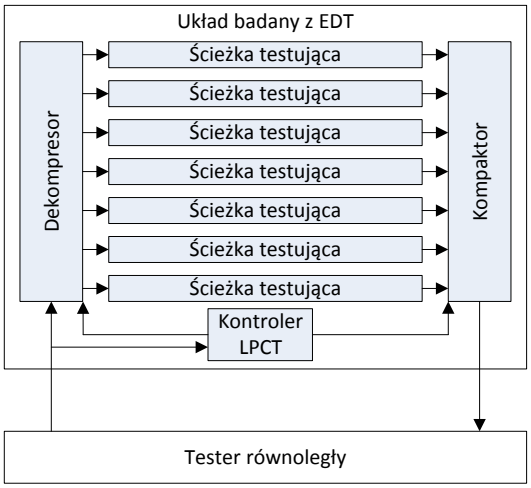
Wśród projektantów urządzeń elektronicznych dominuje obecnie tendencja zamykania jak największej liczby obwodów wewnątrz struktur scalonych, co w oczywisty sposób przyczynia się do miniaturyzacji produktów końcowych i zwiększania ich niezawodności, dzięki eliminacji wielu połączeń stykowych i lutowanych. Ponadto, dominuje tendencja do wyposażania struktur scalonych w ekstremalnie niską liczbę wyprowadzeń zewnętrznych z uwagi na to, że układy te komunikują się „ze światem" za pomocą szybkich, szeregowych układów SerDes (ang. *Serializer-Deserializer* – mianem tym określa się każdy rodzaj interfejsu łącza szeregowego, który po obu stronach udostępnia magistralę równoległą) [2]. Wykorzystanie w układach scalonych łącza SerDes przy jednoczesnym braku innych wyprowadzeń informacyjnych sprawia, że nie można testować ich z wykorzystaniem nawet bardzo wąskiej magistrali równoległej. Ponieważ jednak zawierają one już nadajnik i odbiornik szybkiego łącza szeregowego, stanowi to naturalną alternatywę dla dotychczas wykorzystywanej magistrali równoległej. Obecnie istniejące rozwiązania zakładają jednak wykorzystanie linii testera równoległego do szeregowego przesyłania danych, co znacznie wydłuża czas przebiegu testu. Warto zatem rozważyć zmianę w konstrukcji urządzeń testujących, która umożliwi przeprowadzanie prawdziwie „szeregowego testu" bez wydłużania czasu całej procedury. Niniejsza praca jest propozy-

cją wykorzystania gigabitowych układów SerDes do celów sprzęgania testera z układem scalonym oraz naświetla metodę rozwiązania problemów specyficznych dla tego rodzaju połączenia.

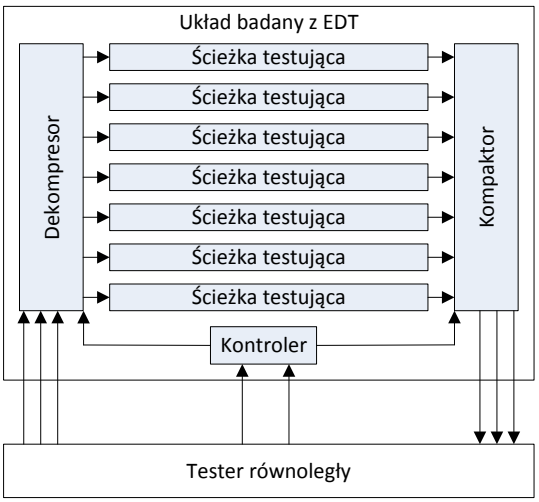
1.2. Przegląd rozwiązań istniejących

Istnieje wiele rozwiązań, których celem jest zmniejszenie liczby linii elektrycznych łączących tester i układ badany. Najprostszym przykładem działania, mającego na celu znaczną redukcję wykorzystywanych linii testera, jest zaproponowane w [3] wykorzystanie zmodyfikowanego interfejsu TAP (ang. *Test Access Port*). TAP, nazywany zwykle JTAG (pomimo, że JTAG oznacza w rzeczywistości grupę ludzi, pracujących nad standardem TAP – ang. *Joint Test Access Group*), jest znanym od lat i szeroko wykorzystywanym interfejsem, zaprojektowanym z myślą o testowaniu krawędziowym, którego celem miało być umożliwienie wykrywania błędnych połączeń lutowanych i stykowych pomiędzy układami scalonymi na płytce drukowanej. Początkowo, zgodnie ze standardem TAP, przerzutniki buforów wejścia/wyjścia układów scalonych łączone były podczas testu w długi rejestr, zwany ścieżką krawędziową. Z czasem zaczęto do ścieżki krawędziowej dołączać także część struktur wewnętrznych. Znanym tego przykładem jest mikrokontroler ATmega 32 firmy Atmel [4], w którym w szereg ścieżki krawędziowej włączono również przetwornik A/C, układy zegarowe, łącze szeregowe (USART) i wiele innych struktur wbudowanych, a także rejestr programatora, dlatego możliwe jest pełne zarządzanie (programowanie, testowanie, monitorowanie) mikrokontrolerem przy wykorzystaniu wyłącznie TAP. Przedstawiony trend do rozszerzania ścieżki krawędziowej o dodatkowe podukłady wykorzystali autorzy wskazanego artykułu [3] proponując, by ścieżka krawędziowa była podzielona na krótsze ścieżki testujące o możliwie równej długości, obejmujące wszystkie (lub prawie wszystkie) przerzutniki, wchodzące w skład testowanego układu. TAP byłby sterowany bezpośrednio z portu testera, bez potrzeby dokonywania jakichkolwiek jego fizycznych modyfikacji. Takie rozwiązanie, choć potencjalnie skuteczne i łatwe w realizacji, ma jedną zasadniczą wadę: czas przeprowadzania testu wzrasta kilkadziesiąt lub kilkaset razy. Powodem tego jest fakt, że linie portu testera przystosowane są do transmisji równoległej i mogą pracować z częstotliwością co najwyżej 50 ÷ 100 (rzadziej 200) MHz, a w każdym takcie zegara wykorzystywanych jest kilkadziesiąt razy mniej linii, niż podczas testu równoległego. Jest to główny powód, dla którego przedstawione rozwiązanie jest niepraktyczne. Podobne założenie (tj. opisane już

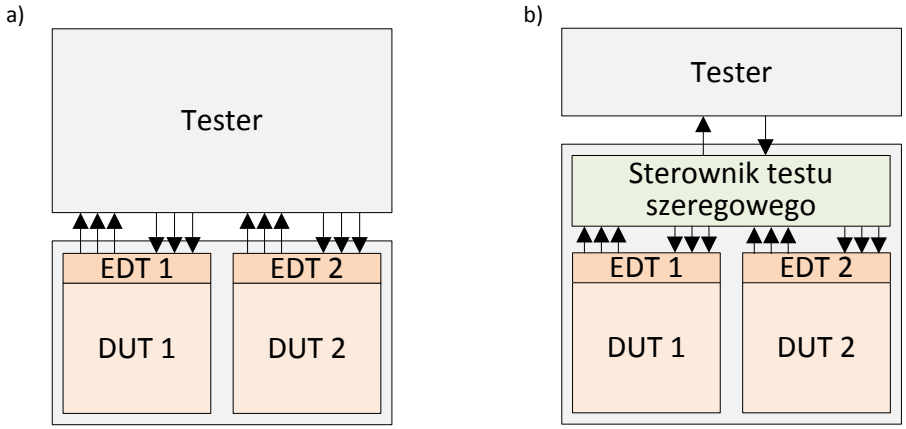
wykorzystanie TAP) przyjęli autorzy artykułu [5]. Zaproponowali oni ponadto, by TAP mógł pracować w trybie połączenia wszystkich ścieżek testujących w jeden łańcuch. Innymi słowy, autorzy zaproponowali by rozbudowaną ścieżkę krawędziową nazywać ścieżką testującą. Ponadto, w artykule proponuje się uzupełnianie przerzutników i interfejsu TAP o dodatkowe układy kombinacyjne, umożliwiające sterowanie mocą pobieraną przez układ badany podczas testu. To rozwiązanie także cechuje się niską szybkością pracy i również nie jest wykorzystywane w praktyce. Ideę testu szeregowego z wykorzystaniem testera równoległego przedstawiono na rysunku 1.1, rysunek 1.2 ukazuje zaś, dla porównania, ideę testu równoległego. Znaczenie wykorzystanego w rysunkach bloku EDT (ang. *Embedded Deterministic Test*) zostanie szerzej wyjaśnione w rozdziale 3.1. Z kolei na rysunku 1.3 pokazano ideowe porównanie techniki jednoczesnego testu wielu struktur (DUT) zawartych w jednym układzie scalonym z wykorzystaniem transmisji równoległej (a) i szeregowej (b).



Rys. 1.1. Testowanie szeregowe z wykorzystaniem testera równoległego. Kontroler LPCT (ang. Low Pin-Count Test) pełni rolę generatora sygnałów sterujących przebiegiem testu na podstawie licznika danych odebranych z testera



Rys. 1.2. Testowanie równoległe



Rys. 1.3. Porównanie idei testu równoległego (a) i szeregowego (b) wielu oddzielnych struktur (DUT) zawartych w jednym układzie scalonym

Nieco lepsze rozwiązanie zaproponowano w publikacji [6]. Zaprezentowano tam kompletną architekturę układu szeregowo-równoległego, pozwalającego na zasilanie kilku równoległych wejść dekompresora danych testowych wykorzystując zaledwie jeden kanał testera. Odpowiedzi układu badanego są odbierane przez tester również przy wykorzystaniu jednego kanału. Wadą tego rozwiązania jest jednak fakt, że dane w kanale transmisyjnym pomiędzy testerem a DUT nie ulegają zwielokrotnieniu w czasie, dlatego prędkość przeprowadzania procedury testowej znacznie spada w stosunku do szybkości tej samej procedury, przeprowadzanej z wykorzystaniem równoległego połączenia ATE – DUT.

Ciekawe rozwiązanie przedstawiono także w artykule [7]. Proponuje się tam przeprowadzanie testu za pomocą popularnej magistrali I^2C. Głównym zadaniem interfejsu I^2C w tym układzie jest dokonywanie konwersji szeregowo-równoległej. Ponadto, w celu

zapewnienia poprawnej pracy sterownika testu w DUT należy wykorzystać jeszcze jeden – trzeci kanał testera, za pośrednictwem którego do DUT przesyłany ma być sygnał taktujący, niezależny od zegara generowanego przez sterownik magistrali I^2C. Zastosowanie tego łatwego w realizacji rozwiązania znacznie wydłuży jednak czas testu produkcyjnego i z tego względu może nie cieszyć się dużą popularnością.

Innym tokiem rozumowania wykazali się autorzy artykułu [8], proponując dodatkową kompresję danych testowych. Kompresja ta miałaby odbywać się na poziomie kodowania kanałowego, byłaby więc niezależna od stosowanej metody testowania (TAP, test równoległy, czy technologia TestKompress firmy Mentor Graphics). Przedstawione wyniki eksperymentów nie są zachęcające. Kompresja ta, np. kodowanie Huffmana, nie jest efektywna, wymaga uzupełniania układu badanego w specjalizowany dekoder kanałowy, a tester musiałby być wyposażony w konfigurowalny koder. To drogie i mało efektywne rozwiązanie nie cieszy się dużą popularnością.

Jedno z lepszych, zdaniem autora pracy, rozwiązań zapewniających zwielokrotnienie czasowe danych na łączu ATE – DUT zaproponowano w artykule [9]. Mimo, że i tu proponuje się wykorzystanie cyfrowych wyprowadzeń testera, mogących pracować z częstotliwością najwyżej kilkuset megaherców, prędkość bitowa w kanale transmisyjnym jest większa, niż częstotliwość taktowania sterownika testu. Propozycja ta jest najbliższa pod względem technicznym połączeniom klasy SerDes.

Wspomniane publikacje ukazują sposoby zmniejszenia liczby linii łączących tester z układem badanym poprzez wykorzystanie magistrali szeregowej, ogólnodostępne w czasie pisania niniejszej pracy. Żadne z wymienionych rozwiązań transmisji szeregowej nie umożliwia wykorzystania pełnej szybkości testu – w każdym z nich wykorzystuje się istniejące, niskoczęstotliwościowe linie testera, a naturalnym jest, że chcąc przesłać szeregowo zbiór danych równoległych, łącze szeregowe musi pracować dużo szybciej, niż emulowana przezeń magistrala równoległa.

1.3. Cel i zakres pracy

Celem pracy jest zaproponowanie nowego rodzaju magistrali wraz z interfejsem i protokołem transmisyjnym, pozwalającej na przeprowadzenie testu produkcyjnego z wykorzystaniem minimalnej liczby połączeń elektrycznych, przy zachowaniu parametrów testu przynajmniej nie gorszych od tych, które możliwe są do osiągnięcia podczas testu z wykorzystaniem tradycyjnej magistrali równoległej. Ponadto, złożoność sprzętowa

interfejsu komunikacyjnego powinna być jak najmniejsza, a sam interfejs, w miarę możliwości, nie powinien wymuszać konieczności ingerencji w strukturę logiki sterującej procesem testowania, zintegrowanej z układem badanym i powinien wykorzystywać implementowane obecnie niemal w każdym układzie bloki szybkiej transmisji danych SerDes. Przedmiotem pracy jest zatem:

- zbadanie parametrów pracy układów podczas przeprowadzania testu z użyciem magistrali równoległej,
- opracowanie fizycznej warstwy szeregowego łącza komunikacyjnego między testerem a układem badanym, wraz z interfejsami końcowymi i sterownikiem magistrali,
- opracowanie teoretyczne, implementacja oraz badanie funkcjonowania protokołu umożliwiającego przeprowadzenie testu z wykorzystaniem opracowanej wcześniej magistrali szeregowej.

Wszystkie potrzebne elementy i bloki funkcjonalne mają zostać zaimplementowane w strukturze FPGA (ang. *Field-Programmable Gate Array*) układu Virtex 6 [10] firmy Xilinx.

2. Testowanie

2.1. Procedura testowania

Procedura testowania cyfrowego układu scalonego rozpoczyna się już podczas jego projektowania. W ostatniej fazie projektu, odpowiednie narzędzia dokonują analizy plików źródłowych, zawierających opis struktury układu (np. w języku Verilog, czy VHDL). W tej fazie kontrolowana jest zgodność projektu z regułami DFT (ang. *Design For Testability* – projektowanie ułatwiające testowanie). Następnie do projektu wstawiane są dodatkowe elementy odpowiedzialne za kontrolę testu, a elementy sekwencyjne (przerzutniki) uzupełniane są dodatkową logiką na wejściach informacyjnych i grupowane w ścieżki testujące (ang. *scan chains*). Kolejnym etapem jest wygenerowanie wektorów testowych dla pokrycia założonego zbioru uszkodzeń. Wyznaczone podczas tego procesu wektory uzupełniane są o dodatkowe informacje, niezbędne testerowi do uruchomienia procedury testowej, na przykład opisy sterowania procedurą zatrzaskiwania danych w ścieżkach testujących (ang. *capture*), czy jawny zapis taktów sygnału zegarowego.

Istnieje wiele formatów plików, przeznaczonych do opisu i przechowywania informacji dotyczących sterowania wyjściami testera. Oto przykłady kilku z nich:

- ASCII – format użyteczny do wzrokowej analizy sygnałów, ponieważ dane zapisane są w bardzo czytelny (dla człowieka) sposób. Utrudnione jest natomiast automatyczne przetwarzanie zawartości plików zapisanych w tym formacie;
- BIN – format binarny, w którym w sposób bezpośredni, w postaci bitów, odwzorowana jest zawartość pamięci testera. Plik w takim formacie nie wymaga wstępnego przetwarzania i może zostać bezpośrednio wczytany przez tester. Utrudnia to jednak przeprowadzanie eksperymentów z wykorzystaniem urządzeń i oprogramowania innych, niż dostarczone przez producenta testera;
- STIL (ang. *Standard Test Interface Language*) – skryptowy opis procedury testowej, w którym wektory testowe zapisane są w sposób jawny, a sposób sterowania transmisją tych danych z testera do układu badanego opisany jest w formie procedur;
- WGL (ang. *Waveform Generation Language*) – nie zawierający makr ani procedur format, w którym każdy bit i każda zmiana poziomu sygnału jest zapisana wprost, a każda linia wyjściowa lub wejściowa testera ma na stałe przypisaną pozycję w kolejnych wierszach pliku.

Niezależnie od formatu zapisu, wygenerowane dane, w toku dalszych czynności, analizowane są przez oprogramowanie testera i zapisywane w jego wewnętrznej pamięci. Tester zaś, za pomocą specjalnej głowicy stykowej, łączony jest bezpośrednio z układem badanym, nie wyposażonym jeszcze w obudowę, oraz przeprowadza test, zgodnie z zapisanymi w jego pamięci informacjami. Po przesłaniu każdego wektora testowego dane odebrane od DUT są porównywane z zapisanymi uprzednio w pamięci testera wartościami oczekiwanymi. Negatywny wynik porównania oznacza, że testowany układ zawiera jedno lub więcej uszkodzeń. Na podstawie analizy wszystkich odpowiedzi możliwe jest często ustalenie rodzaju i miejsca powstania uszkodzenia. Statystyczna analiza uszkodzeń dużej grupy układów może również ujawnić nieprawidłowości w procesie produkcyjnym. Celem testu jest więc nie tylko eliminacja pojedynczych uszkodzonych układów scalonych, lecz także monitorowanie całego procesu produkcyjnego.

2.2. Testowanie równoległe

Historycznie pierwszym i do dziś stosowanym interfejsem łączącym ATE (ang. *Automatic Test Equipment* – tester wraz z interfejsem komunikacyjnym) i DUT jest sprzęg równoległy. W zależności od złożoności DUT, może występować od kilkudziesięciu do kilkuset równoległych połączeń elektrycznych. Z uwagi na wspomnianą złożoność, przy jednoczesnej tendencji do zmniejszania liczby wyprowadzeń elektrycznych obudowanego układu scalonego wskazane jest, aby liczba linii niezbędna do przeprowadzenia testu jednego DUT była jak najmniejsza, co bezpośrednio przełoży się na koszt produktu dla użytkownika końcowego.

W strukturze równoległej magistrali ATE – DUT, znajdują się wydzielone linie wejściowe i wyjściowe ścieżek testujących układu badanego, linie zegara (ich liczba zależy między innymi od złożoności DUT), linie wejść pierwotnych (tzn. nie objętych ścieżkami testującymi) i linie kontrolne przeznaczone dla celów komunikacji z umieszczoną w DUT logiką sterującą testem. Prędkość transmisji zależy od liczby linii danych i częstotliwości sygnału strobującego zapis do ścieżek testujących, która zazwyczaj wynosi 25 lub 50 MHz.

2.3. Testowanie szeregowe

Obecnie zakłada się prowadzenie transmisji szeregowej między ATE a DUT za pomocą wybranych linii istniejącej magistrali równoległej. Zaletą takiej realizacji transmisji

szeregowej jest prostota i brak potrzeby wprowadzania zmian sprzętowych w układzie testera, jednak znaczne spowolnienie transferu danych jest poważną wadą tego rozwiązania. Spowolnienie to jest spowodowane faktem, iż każda linia portu wejść/wyjść testera, z założenia mająca stanowić pojedynczy element sprzęgu równoległego, cechuje się niską maksymalną częstotliwością pracy. Częstotliwość maksymalna zwykle wynosi 100 MHz. Jedynie nieliczne testery posiadają wyjścia równoległe, mogące pracować z szybkością dochodzącą do 200 MHz.

W celu realizacji testu z użyciem małej liczby połączeń elektrycznych, oraz by uniknąć konieczności stosowania redundantnych połączeń linii sterujących, po stronie odbiorczej (DUT) alokuje się kontroler LPCT (ang. *Low-Pin Count Test*), którego zadaniem jest odtwarzanie sygnałów sterujących przebiegiem procedury testowej (np. sygnału Update, powodującego zatrzaśnięcie odpowiedzi układu na wprowadzone doń uprzednio, w postaci wektora testowego, wymuszenie). Wspomniany kontroler LPCT stanowi obecnie nieodłączną część sprzętowego kontrolera testu szeregowego i utrudnia lub uniemożliwia wykorzystanie wszystkich funkcjonalności, oferowanych przez narzędzia przeznaczone do generacji testów.

3. Protokół EDT-SerDes

3.1. Informacje wstępne

W podrozdziale 2.3 opisano zalety i wady obecnie realizowanego testu szeregowego. Niniejsza praca jest propozycją wyeliminowania wad, jakimi w mniemaniu jej autora są: niska prędkość transmisji danych oraz konieczność stosowania kontrolera LPCT, a także niemożność wykorzystania wszystkich opcji narzędzi generujących wektory testowe.

Pomysłem autora pracy, wyróżniającym przedstawione poniżej rozwiązanie wśród wszystkich opisanych dotąd koncepcji testu szeregowego, jest przede wszystkim uczynienie sprzęgu transmisji szeregowej całkowicie transparentnym z punktu widzenia sterownika testu umieszczonego w układzie badanym (DUT). Ponadto, w pracy proponuje się wykorzystanie innego, niż dotychczas, interfejsu komunikacyjnego – z grupy tzw. układów SerDes z szybkimi wyjściami/wejściami cyfrowymi. Układem SerDes można nazwać każdy kompletny sterownik transmisji szeregowej, który po obu stronach łącza (z punktu widzenia nadajnika i odbiornika transmisji) udostępnia magistralę równoległą. Szczególnym przykładem układu SerDes jest stosowany powszechnie w komputerach osobistych sprzęg urządzeń magazynujących Serial-ATA, lub sprzęg kart rozszerzeń komputera PC z magistralą PCI-Express. Obydwa wspomniane sprzęgi wykorzystują szybkie nadajniki/odbiorniki, prowadzące transmisję w kanale przewodowym elektrycznym. Transmisja ta, z punktu widzenia kanału elektrycznego, ma charakter analogowy. Niniejsza praca również jest propozycją wykorzystania tego rodzaju transmisji do komunikacji między testerem a układem badanym, ze względu na możliwą gigabitową szybkość transferu danych oraz bardzo powszechne występowanie kontrolerów SerDes w strukturach większości współczesnych układów scalonych wielkiej skali integracji (VLSI – ang. *Very Large Scale of Integration*).

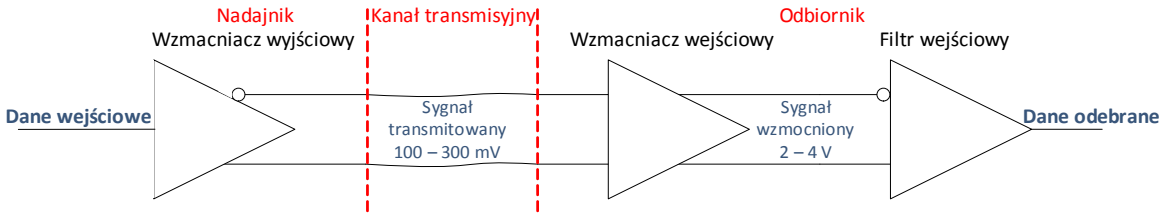
Wyjaśnienia wymaga widoczna w tytule bieżącego rozdziału nazwa proponowanego protokołu transmisyjnego – „EDT-SerDes". Na szczególną uwagę zasługuje zastosowany w niej akronim EDT [11] (ang. *Embedded Deterministic Test*). Jest to nazwa technologii przeprowadzania testu produkcyjnego układów scalonych, opracowanej i wykorzystywanej przez firmę Mentor Graphics. Głównymi założeniami tej technologii są: redukcja czasu przeprowadzania testu i znaczne zmniejszenie zapotrzebowania na pamięć testera (kompresja danych testowych). Szeregowe łącze, będące przedmiotem niniejszej pracy, zoptymalizowane jest do współpracy właśnie z układami testowanymi

z wykorzystaniem technologii EDT, choć może zostać wykorzystane również podczas testu z wykorzystaniem innych technologii testowania.

3.2. Warstwa sprzętowa

Część sprzętowa (elektryczna) łącza SerDes nie jest jednoznacznie zdefiniowana. Na potrzeby testowania należy jednak przyjąć pewne praktyczne założenia, pozwalające zrealizować opisany w kolejnych rozdziałach protokół EDT-SerDes.

Najważniejszym założeniem jest transmisja w paśmie podstawowym. Ponieważ częstotliwość modulacji ma osiągać wartości powyżej 1 GHz, kanał transmisyjny powinien pracować w trybie różnicowym z wydzielonym ekranem – dla transmisji na odległość większą niż kilka centymetrów między sterownikiem, a głowicą testera, lub bez ekranu – w przypadku transmisji między głowicą testera a układem badanym. Odbiornik SerDes powinien być wyposażony w dolnoprzepustowy wzmacniacz, o 3-decybelowym paśmie nieco tylko wyższym od częstotliwości modulacji sygnału transmitowanego. Uproszczony schemat blokowy układu łącza SerDes, niezbędnego przy realizacji protokołu EDT-SerDes, pokazany jest na rysunku 3.1.



Rys. 3.1. Uproszczony schemat blokowy części sprzętowej łącza EDT-SerDes

Innym ważnym założeniem EDT-SerDes jest fakt, że protokół komunikacyjny nie jest trwale powiązany ze sprzętową realizacją nadajnika i odbiornika, nie ma bowiem żadnych wymagań elektrycznych ponad te, które zostały wymienione. Jest to szczególnie istotne dla konstruktorów układów scalonych, które będą testowane z wykorzystaniem SerDes, ponieważ nie będą oni zmuszani do rygorystycznego przestrzegania norm sprzętowych, a co za tym idzie, na cennej powierzchni krzemowej układu scalonego oszczędzą być może odrobinę miejsca, które musiałoby być przeznaczone na implementację układu transmitera innego niż te zastosowane już w aplikacji.

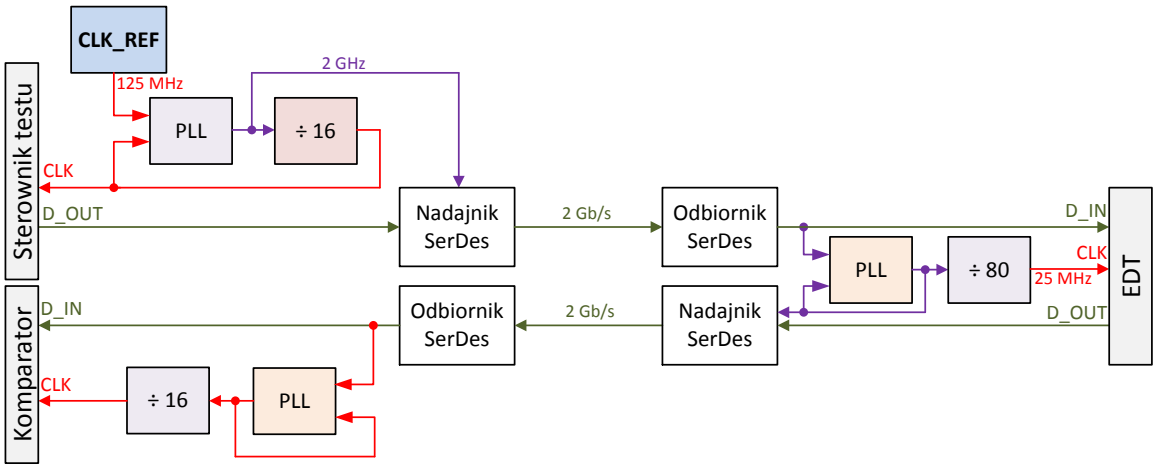
Fakt nieścisłego podania parametrów sprzętowych może być jednak sporym utrudnieniem dla projektantów testerów, wspierających transmisję EDT-SerDes, ponieważ muszą oni tak zaprojektować tester, by z dużym prawdopodobieństwem spełnił oczekiwania jak najszerszego grona producentów układów scalonych.

Należy więc zaznaczyć, że układy nadawczy i odbiorczy testera powinny spełniać przynajmniej poniższe wymagania:

- regulowana impedancja wyjściowa/wejściowa, przynajmniej w zakresie 25 ÷ 120 Ω, z krokiem co najmniej 10Ω, ponieważ w praktyce spotyka się najczęściej układy szybkich łączy szeregowych o impedancjach 50, 75, oraz 100 i 120 Ω;
- regulowany poziom napięcia sygnału wyjściowego nadajnika, przynajmniej w zakresie 100 ÷ 1000 mV z krokiem co najmniej 50 mV, ponieważ jak wykazały próby (opisane w rozdziałach kolejnych) minimum funkcji stopy błędu w zależności od amplitudy sygnału transmitowanego mieści się zwykle w granicach 300 ÷ 500 mV;
- regulowana czułość wzmacniacza wejściowego, w zakresie przynajmniej 50 ÷ 1000 mV, z krokiem przynajmniej 50 mV, z uwagi na konieczność dopasowania do wspomnianego wyżej zakresu napięć wzmacniacza wyjściowego, z uwzględnieniem możliwej tłumienności przewodów;
- regulowana częstotliwość modulacji sygnału transmitowanego (w paśmie podstawowym), przynajmniej w zakresie 1 ÷ 4 GHz, z uwagi na zachowanie zgodności z narzędziami testującymi i założeniami EDT, oraz typową częstotliwością sygnału taktującego sterownik testu wynoszącą około 50 MHz, z zachowaniem stosownego zapasu, umożliwiającego przeprowadzanie testu z częstotliwością 200 MHz za pomocą 50 kanałów równoległych.

W założeniu transmisja EDT-SerDes ma charakter synchroniczny. Oznacza to, że jedynym źródłem sygnału podstawy czasu w systemie testującym jest tester. Układ badany dostraja częstotliwość swojego zegara do częstotliwości zegara testera, poprzez odzysk sygnału taktowania z sygnału odebranego za pomocą łącza SerDes. Ten sam odzyskany sygnał służy także do taktowania logiki EDT i układu testowanego, chyba, że podczas testu wykorzystywana jest technologia *At-Speed* (test z pełną szybkością). W takim przypadku, za generowanie sygnału taktującego układ testowany, odpowiada odrębny kontroler OCC [12] (ang. *On-chip Clock Controller*), współpracujący z pętlą fazową, będącą częścią układu badanego. Kontroler OCC nie jest jednak częścią EDT-SerDes, a jego wykorzystywanie nie wymaga żadnych dodatkowych konfiguracji łącza szeregowego, co zostanie dokładniej objaśnione w dalszej części pracy.

Sposób propagacji sygnału podstawy czasu przedstawiony jest na rysunku 3.2, wraz z przykładowymi wartościami liczbowymi. Łatwo zauważyć, że w całym systemie występuje tylko jedno źródło sygnału odniesienia (CLK_REF), umieszczone po stronie nadawczej testera. Sygnał zegarowy generowany przez to źródło jest zwielokrotniany za pomocą pętli fazowej. Zwielokrotniony do częstotliwości pracy łącza SerDes sygnał jest przekazywany do nadajnika SerDes oraz dzielnika częstotliwości, wyjście którego pełni rolę sprzężenia zwrotnego pętli fazowej, a także źródła podstawy czasu dla sterownika przebiegu testu (w szczególności dla pamięci, w której przechowywane są wektory testowe). W tym miejscu należy zaznaczyć, że pamięć testera musi być podzielona na dwa fizycznie odrębne bloki. Pamięć wartości oczekiwanych i układ komparacji będą taktowane sygnałem odzyskanym z EDT-SerDes, w celu zachowania pełnej synchronizacji w procesie testowania. To dość nietypowe założenie wyeliminuje konieczność stosowania buforów i potrzebę wstawiania danych dopełniających, charakterystycznych dla transmisji częściowo plezjochronicznej – nie można bowiem zapominać, że każda pętla fazowa zastosowana w układzie wnosi dodatkowe zniekształcenia wolnozmienne (ang. *wander*) i nie można zagwarantować, że dane odebrane od EDT będą w wystarczającym synchronizmie z danymi wysłanymi z testera.

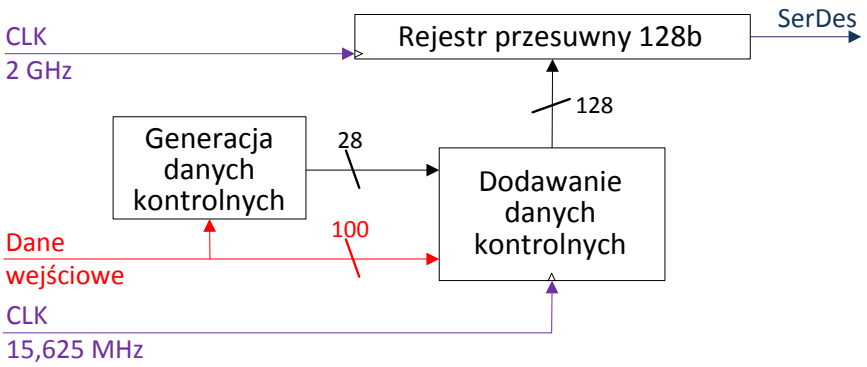


Rys. 3.2. Schemat propagacji sygnału podstawy czasu i synchronizacji

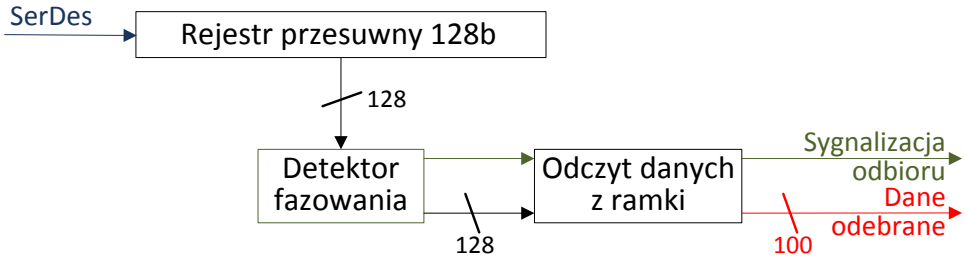
3.3. Warstwa logiczna

Rysunek 3.3 ukazuje schematyczny obraz struktury nadajnika EDT-SerDes. Dane liczbowe stanowią jedynie przykład. Główny rejestr przesuwny taktowany jest sygnałem o częstotliwości pracy łącza SerDes. Jest to rejestr równoległo-szeregowy – dane zapisy-

wane są doń magistralą równoległą, a odczytywane szeregowo. Odczytane z tego rejestru kolejne bity są bezpośrednio przekazywane do części analogowej łącza, skąd po procesie modulacji przekazywane są do warstwy fizycznej łącza. Źródłem danych rejestru przesuwnego jest układ, którego zadaniem jest uzupełnianie „surowych" danych wejściowych, stanowiących w istocie zapis stanu kanałów testera, obserwowanych w chwili próbkowania; próbkowanie to z kolei przeprowadzane jest w tym przypadku z częstotliwością co najwyżej 15,625 MHz. Podana częstotliwość wynika z szybkości pracy łącza i długości ramki transmisyjnej: 2 GHz / 128 = 15,625 MHz.. Rodzaj danych kontrolnych i sygnalizacyjnych szerzej omówiony będzie w rozdziale 3.4, należy jednak w tym miejscu wspomnieć, że na początku ramka uzupełniana jest o 8-bitową sekwencję startową (fazującą). Sekwencja ta jest wykorzystywana przez odbiornik. Schematyczną strukturę odbiornika przedstawia rysunek 3.4. W każdym takcie zegara testowany jest stan 8 najmniej znaczących bitów rejestru wejściowego za pomocą układu nazwanego na rysunku detektorem fazowania. Wbudowany weń komparator generuje na wyjściu stan aktywny, jeśli testowane bity wejściowe są zgodne z prawidłową sekwencją fazującą.



Rys. 3.3. Struktura logiczna nadajnika EDT-SerDes



Rys. 3.4. Struktura logiczna odbiornika EDT-SerDes

Stan wyjścia komparatora jednocześnie stanowi informację o odbiorze nowych danych, która traktowana jest jako źródło sygnału odniesienia (taktowania), wykorzystywanego przez EDT i pozostałe struktury układu badanego. Warto zauważyć, że sygnał odbioru nowych danych jest zgodny co do częstotliwości z częstością wysyłania kolejnych ramek przez tester. Stanowi to dowód spełnienia założenia, że to tester jest głównym źródłem sygnału taktującego dla układu badanego i nie jest potrzebne stosowanie dodatkowych, zewnętrznych źródeł. Ponadto, ponieważ dane odbierane pozostają w synchronizmie (ze stałym przesunięciem w dziedzinie czasu) z danymi wysyłanymi przez tester, wszystkie sygnały sterujące przebiegiem testu mogą być przesyłane za pomocą łącza EDT-SerDes. Nie ma potrzeby stosowania dodatkowych układów odzyskujących lub wyznaczających stan linii kontrolnych, zintegrowanych z EDT (w szczególności LPCT), których istnienie, proponowane w rozwiązaniach konkurencyjnych (opisanych w rozdziale 1.2), ograniczałoby zakres zastosowań komunikacji szeregowej i uniemożliwiało wykorzystanie pełni możliwości narzędzi generujących wektory testowe. Biorąc pod uwagę wyżej opisaną strukturę logiczną układu EDT-SerDes łatwo zauważyć, że cel, jakim było zachowanie jego transparentności dla DUT, został osiągnięty.

3.4. Protokół komunikacji

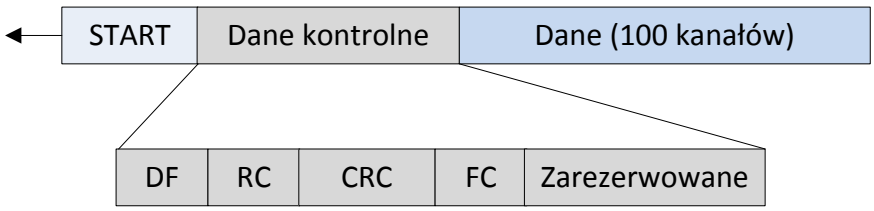
Protokół komunikacji EDT-SerDes został przewidziany specjalnie do celów testowania i z tego względu umożliwia realizację wszystkich możliwości, jakie oferują narzędzia do generacji testów firmy Mentor Graphics, w szczególności:

- zapewnia realizację opcji równoczesnego testowania dowolnej liczby układów (np. rdzeni układu scalonego), przy czym fazy testu tych układów nie muszą być ze sobą skorelowane,
- sterownik EDT-SerDes umieszczony przy układzie badanym dostarcza do logiki EDT sygnał zegarowy, skorelowany z sygnałem podstawy czasu testera (tylko od testera zależy, z jaką szybkością będzie przebiegać test i z jaką szybkością dane będą wprowadzane do ścieżek testujących),

- jest w pełni kompatybilny z kontrolerem sygnałów zegarowych (OCC – *On-chip Clock Controller*) [12] stosowanym w technologii *At-speed* (test z pełną szybkością) firmy Mentor Graphics,
- zapewnia detekcję błędów transmisji w obu kierunkach (od i do testera), z możliwością stwierdzenia, przy transmisji w którym kierunku i w której ramce wystąpił błąd.

Konstrukcja ramek transmisyjnych

Jak wspomniano w rozdziale 3.3, „surowe" dane zostają uzupełnione o informacje dodatkowe (dane kontrolne). Podane zostaną wartości przykładowe, dla których przeprowadzono badania funkcjonowania łącza i które nie są obligatoryjne. Rysunek 3.5 przedstawia strukturę ramki transmisyjnej EDT-SerDes. Znaczenie poszczególnych pól ramki wyjaśnione jest w tabeli 3.1. Całkowita długość ramki wynosi 128b.



Rys. 3.5. Struktura ramki transmisyjnej protokołu EDT-SerDes

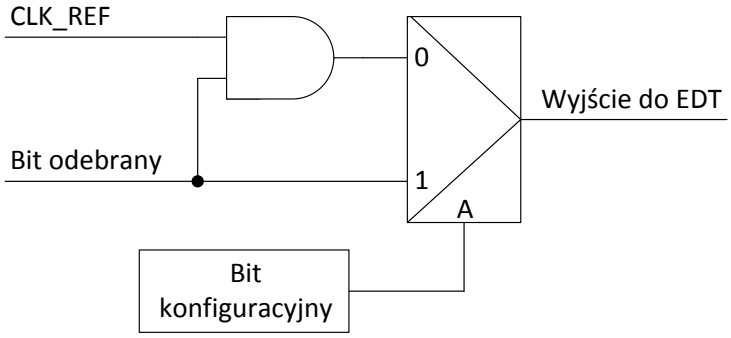
Tabela 3.1. Znaczenie poszczególnych pól ramki transmisyjnej EDT-SerDes

Pole	Długość	Uwagi
START	8b	Sekwencja startowa (fazująca).
DF	1b	*Data Frame*. Znaczenie w zależności od wartości: 0: ramka zawiera dane konfiguracyjne EDT-SerDes 1: ramka zawiera dane (np. wektory testowe).
RC	1b	*Received Correctly*. Bit ten ustawiany jest przez sterownik SerDes w układzie badanym. Znaczenie jest ważne dla testera: 0: ostatnia ramka dotarła do EDT z błędem lub co najmniej jedna ramka w kierunku od testera nie została odebrana, 1: ostatnia ramka dotarła do EDT poprawnie.
CRC	8b	*Cyclic Redundancy Check*. Suma kontrolna. Wyliczana metodą kombinacyjną, dla zwiększenia szybkości pracy układów SerDes. Suma kontrolna wyznaczana jest z całej ramki, łącznie z danymi kontrolnymi (z wyłączeniem sekwencji fazującej i pola CRC).
FC	4b	*Frame Counter*. Cykliczny licznik ramek. Pozwala stwierdzić, że określona liczba ramek (maks. 15) została „zagubiona".
Zarezerwowane	6b	Pole niewykorzystane w obecnej wersji EDT-SerDes.
Dane	100b	Pole danych. Jeśli DF=0, pole to służy to transmisji danych sterujących SerDes, w przeciwnym przypadku w polu tym zawarte są dane testowe.

Bramkowanie zegara

W standardowym testerze z portem równoległym, linie wyjściowe są konfigurowalne. Wybrane z nich mogą pełnić rolę linii danych, inne – linii zegarowych. Jest elementarna różnica w działaniu obu typów linii. Jeśli wartość bitu odpowiadającego rozpatrywanej linii w kolejnym słowie pamięci testera ma wartość logicznej 1, a linia portu skonfigurowana jest do propagowania sygnału czasu, wówczas na rozważanym wyjściu pojawi się bezpośrednio takt sygnału zegarowego testera. W celu zachowania możliwości konfiguracji wyjść, w ramach protokołu EDT-SerDes wprowadzono ramki konfiguracyjne. W bloku odbiorczym łącza, przy EDT, umieszczony został dodatkowy rejestr konfiguracyjny o długości równej liczbie bitów emulowanej magistrali równoległej (100b). Każda linia tej magistrali ma swój bit konfiguracyjny w tym rejestrze. Każdorazowo po odebraniu kolejnej ramki (DF=1) dane są transmitowane przez układ komórek bramkowania zegara. Zawartość każdej takiej komórki uwidoczniona jest na rysunku 3.6. Na wejście CLK_REF komórki bramkowania zegara może zostać podany sygnał podstawy czasu ze

sterownika EDT-SerDes (przypadek najczęstszy) albo sygnał z dodatkowej pętli fazowej (rozwiązanie opcjonalne, relatywnie trudne do kontrolowania). „Bit konfiguracyjny" jest odpowiadającym danej linii bitem wspomnianego rejestru konfiguracyjnego.



Rys. 3.6. Komórka bramkowania zegara

Przed przystąpieniem do realizacji procedury testowania szeregowego, tester powinien wysłać odpowiednią ramkę konfiguracyjną do EDT. Zawartość tej ramki ustalana jest wprost na podstawie danych wygenerowanych dla testera równoległego i nie jest potrzebne wprowadzanie dodatkowego oprogramowania przetwarzającego. Ponadto, rolą ramki konfiguracyjnej jest także zerowanie układu zliczającego ramki (pracującego w oparciu o zawartość pola FC). Dla ustrzeżenia się przed możliwymi błędami transmisji, zaleca się kilkakrotne przesłanie ramki konfiguracyjnej. Wyjścia EDT-SerDes można rekonfigurować w dowolnym momencie – jest to kolejna cecha sprawiająca, że proponowany test szeregowy może obsługiwać rozbudowane sieci połączeń wielu bloków układu scalonego.

Po dokonaniu konfiguracji linii docierających do logiki EDT można rozpocząć realizację właściwej procedury testowej. Jej przebieg nie jest przedmiotem niniejszej pracy, a algorytm samego testu dla proponowanej konstrukcji łącza nie różni się niczym od algorytmu testu równoległego. Różnica polega jedynie na sposobie komunikacji testera z układem badanym. Prześledźmy zatem, co dzieje się podczas pojedynczego taktu zegara, sterującego wysyłaniem ramek transmisyjnych przez tester:

1. Zbocze dodatnie sygnału zegarowego testera wymusza przesłanie ramki, bit po bicie.
2. Po przesłaniu wszystkich 128 bitów, EDT-SerDes oblicza wartość CRC i porównuje ją z wartością CRC odebranej ramki. Wynik komparacji zachowuje

w rejestrze LRC *(Last Received Correctly* – pamięć stanu znacznika RC z ostatniej odebranej ramki).

3. Niezależnie od wyniku komparacji, jeśli DF=1, dane są transmitowane do wejść EDT, poprzez komórki bramkowania zegara, a stan wyjść EDT jest zatrzaskiwany w buforze nadajnika SerDes układu badanego.
4. W nadajniku SerDes układu badanego formowana jest ramka transmisyjna dla testera, której pole RC zapisywane jest wartością uprzednio zachowanego w rejestrze LRC stanu poprawności ostatnio odebranej ramki. Licznik ramki (zawartość pola FC) jest w tym przypadku kopią pola FC ostatnio odebranej ramki.
5. Uformowana ramka jest przesyłana do testera.
6. Część odbiorcza testera kontroluje stan ostatniej ramki i podejmuje odpowiednią czynność, w zależności od stanu pól kontrolnych. Możliwe czynności zostały spisane w tabeli 3.2.

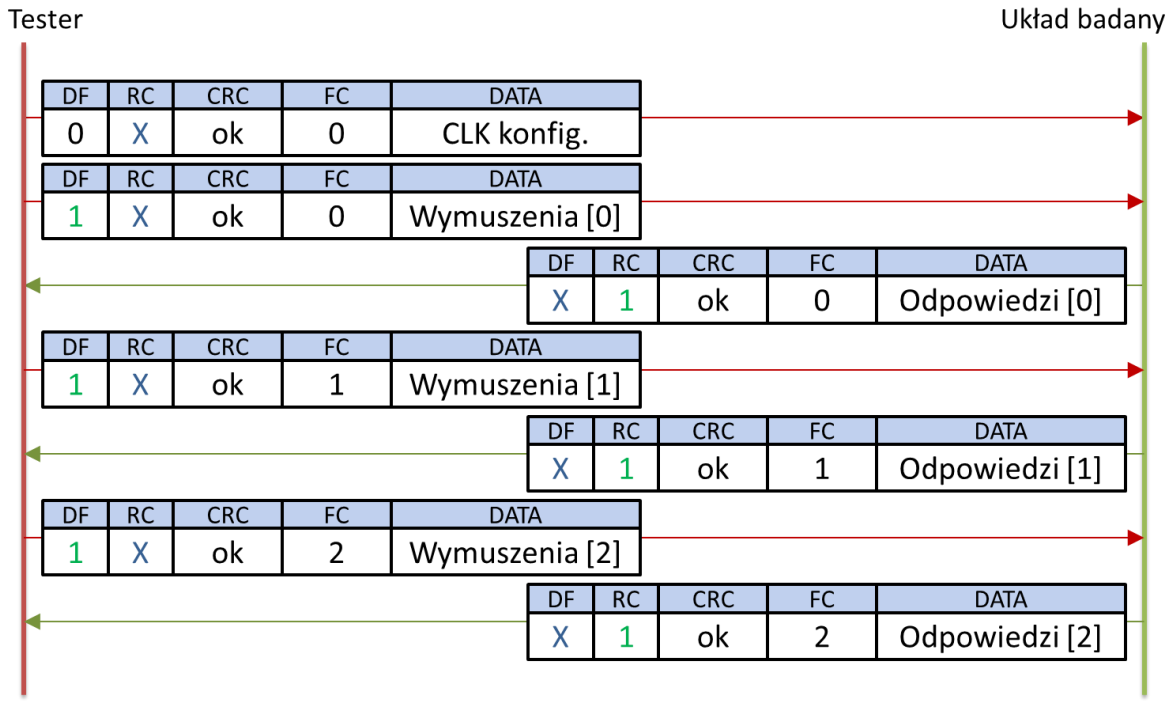
Tabela 3.2. Możliwe czynności podejmowane przez tester po odebraniu ramki od EDT

FC	RC	CRC	Czynność
Kolejny	1	Zgodne	Transmisja bezbłędna; kontynuacja testu.
Kolejny	X	Niezgodne	Ramka uszkodzona; powtórka testu.
Kolejny	0	Zgodne	EDT otrzymał błędną ramkę; powtórka testu.
Niekolejny	X	X	Zgubiona ramka; powtórka testu.

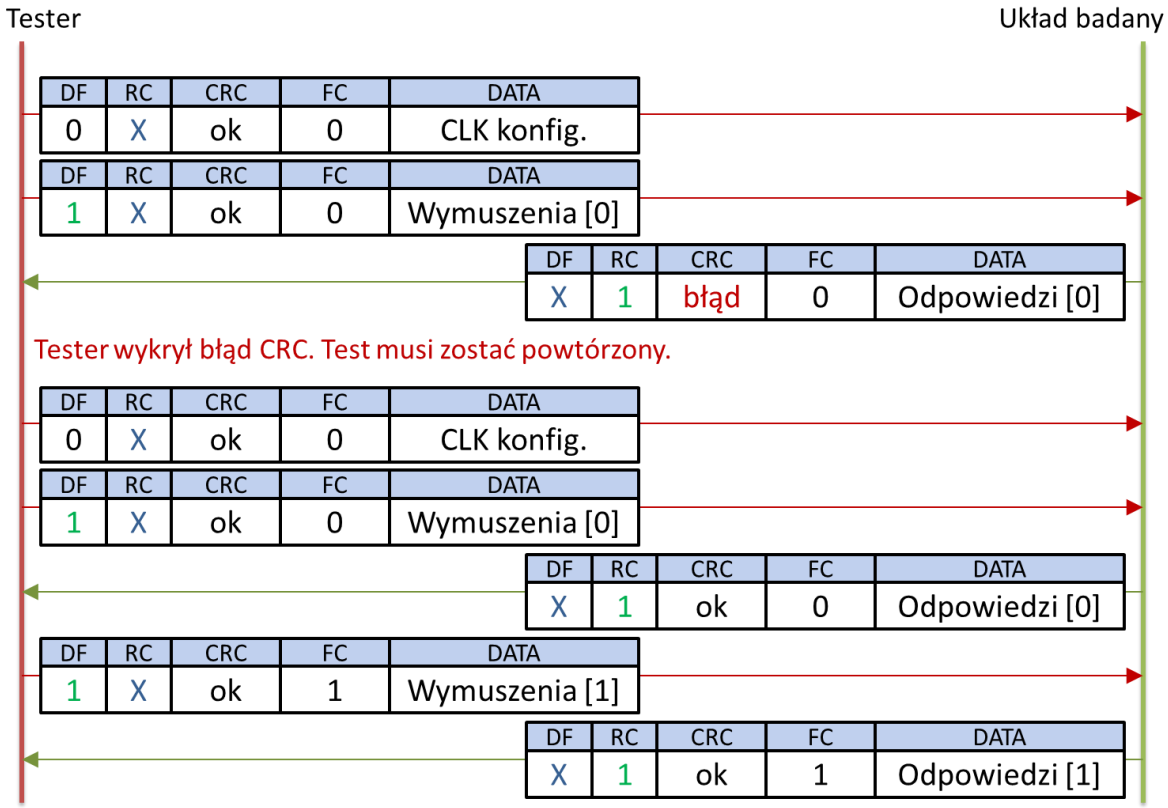
Obsługa błędów

Protokół EDT-SerDes nie umożliwia korekcji błędów. Możliwe są jedynie detekcja, oraz wykrycie miejsca ich wystąpienia. Rysunki od 3.7 do 3.10 ukazują diagramy przepływu informacji dla różnych scenariuszy przebiegu transmisji. Na rysunku 3.7 widać diagram dla sytuacji prawidłowej. Transmisję inicjuje tester, wysyłając kilkakrotnie ramkę konfiguracyjną. Mechanizmy protekcji zaimplementowane w odbiorniku zostają zainicjowane, a pamięć konfiguracji bloku bramkowania zegara zostaje zapisana zawartością pola danych ramki konfiguracyjnej. Odbiór ramki konfiguracyjnej nie skutkuje odesłaniem odpowiedzi do testera. Następnie, tester rozpoczyna transmisję pierwszego wektora testowego, ramka po ramce. W odpowiedzi na każdą ramkę danych odbiornik odsyła do testera odpowiedź, zawierającą dane mechanizmu protekcji i bieżący stan wyjść kompaktora EDT. Choć nie jest to zobrazowane na

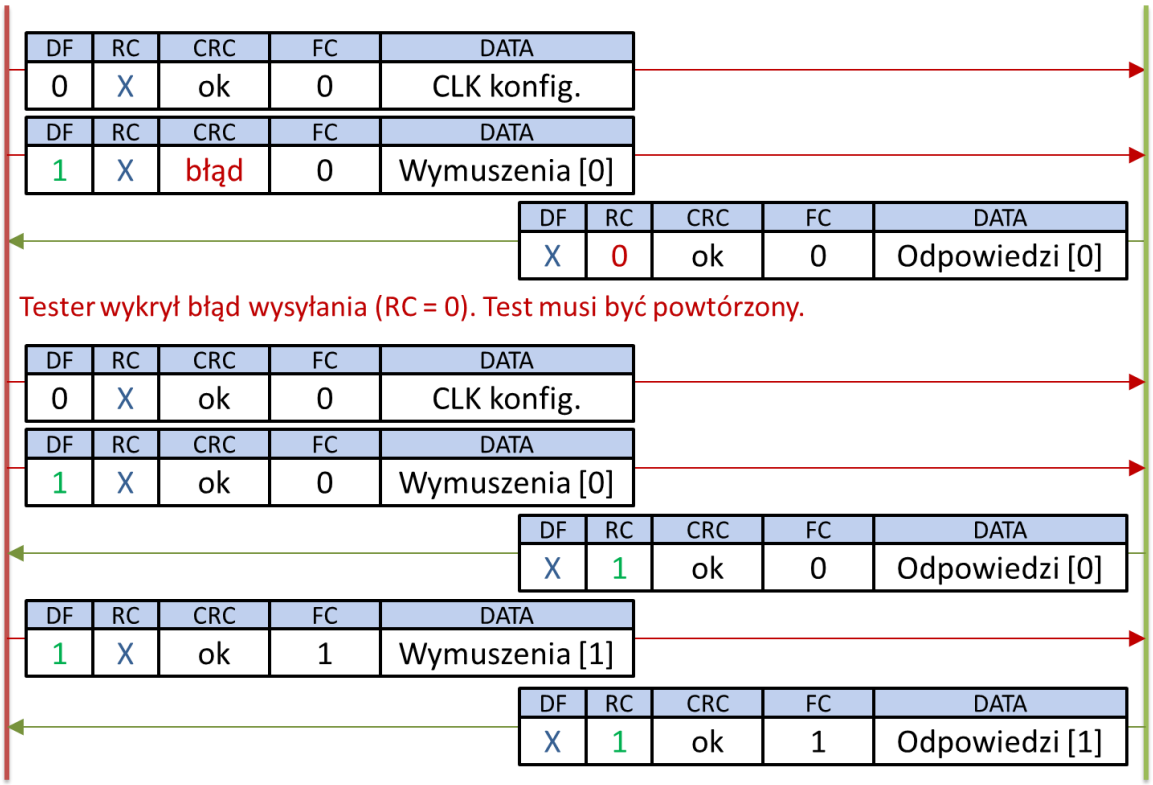
diagramie 3.7, tester nie wstrzymuje nadawania na czas oczekiwania na nadejście odpowiedzi od DUT - nadawanie i odbiór są, z punktu widzenia testera zgodnego z EDT-SerDes, procesami niezależnymi, choć pozostającymi ze sobą w stałej korelacji. Niezależność tych procesów wynika ze stałego w czasie opóźnienia, związanego z fizycznymi cechami połączenia przewodowego ATE i DUT. Pozostałe rysunki (3.8 – 3.10) uwidaczniają sytuacje awaryjne: błędne pole CRC (w kierunku do testera), nieokreślony błąd w kierunku od testera (sygnalizacja w polu RC), oraz sytuację nieodebrania (zagubienia) ramki. Ostatni ze scenariuszy może wystąpić na przykład wówczas, gdy w sekwencji fazującej (startowej) pojawi się błędny bit. Podstawowym mechanizmem detekcji błędów jest 8-bitowa suma kontrolna (CRC). Dodatkowo, ponieważ układ badany nie jest urządzeniem autonomicznym, może on zasygnalizować testerowi, że odebrał błędną ramkę, za pomocą wyzerowanej flagi RC. Ostatnim zabezpieczeniem jest 4-bitowy, cykliczny licznik ramek, którego analiza umożliwia wykrycie maksymalnie 15 zagubionych (nieodebranych) ramek. Każdorazowe wykrycie błędu skutkuje powtórzeniem całej procedury testowej od początku (lub od najbliższego, specjalnie oznaczonego miejsca).



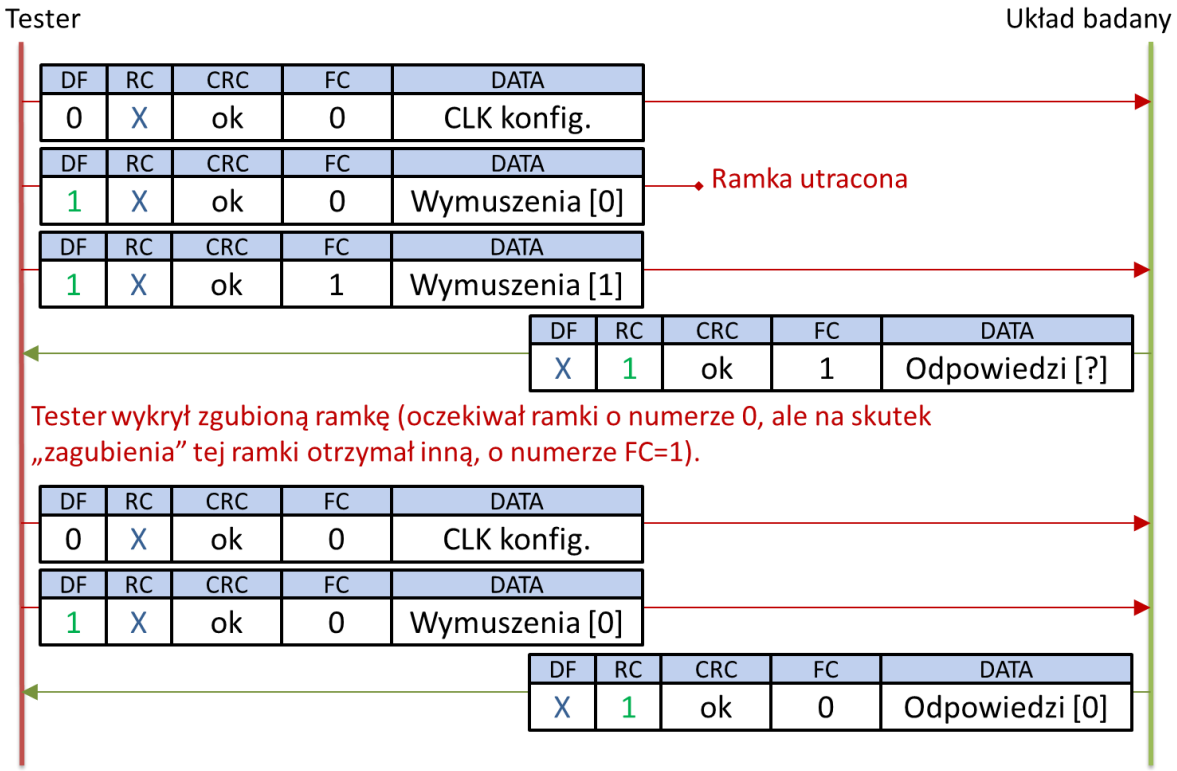
Rys. 3.7. Diagram przepływu informacji dla transmisji bez błędów



Rys. 3.8. Diagram przepływu informacji dla transmisji z błędem (błąd CRC)



Rys. 3.9. Diagram przepływu informacji dla transmisji z błędem w kierunku od testera do EDT



Rys. 3.10. Diagram przepływu informacji w przypadku zaginięcia ramki

4. Dynamiczna alokacja kanałów

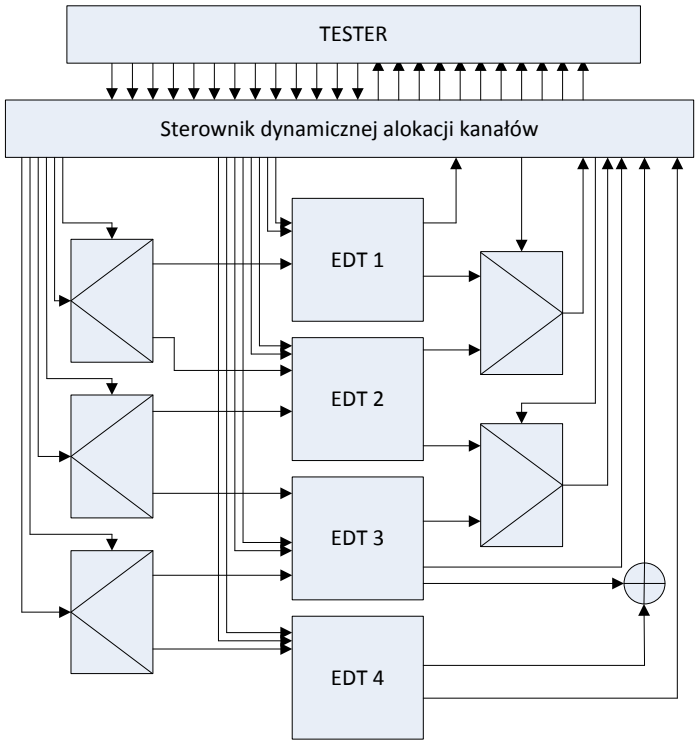
Artykuł [13] opisuje bardzo interesujące zagadnienie, związane z dynamicznym dysponowaniem przydziału kanałów testera do poszczególnych struktur układu scalonego. Struktura jest tu rozumiana jako oddzielny DUT z własną logiką EDT. Można zauważyć, że w związku z kompresją danych, niektóre wektory testowe wymagają wykorzystania większej liczby kanałów niż pozostałe. Normalnie w takiej sytuacji tester z DUT musiał by zostać połączony za pomocą co najmniej tylu kanałów, ile wymagane jest do przesłania wektora testowego który wymaga ich największej liczby. Możliwe jest jednak przełączanie niepotrzebnych w danym momencie kanałów tak, by mogły być one wykorzystane dla potrzeb transmisji danych do innego DUT, który w tej samej chwili wymaga szerszej magistrali do przesłania wektora testowego. Rozwiązanie to pozwala zmniejszyć sumaryczne zapotrzebowanie całego układu scalonego na kanały testera, niezbędne do przeprowadzenie całej procedury testowej wszystkich zawartych w nim DUT. Jest to szczególnie istotne przy korzystaniu z szeregowej transmisji danych za pomocą protokołu EDT-SerDes, pozwala bowiem na równoczesny test większej liczby DUT niż jest to możliwe bez wykorzystania dynamicznej alokacji kanałów. Z tego powodu współpraca obu tych technologii została również sprawdzona, co zostało opisane w rozdziale 6.3.

Rysunek 4.1 przedstawia przykładową strukturę układu SoC (ang. *system-on-chip*) pracującego podczas testu z siecią dynamicznej alokacji kanałów. Warto zwrócić uwagę na to, że niektóre kanały komutowane są za pomocą widocznych multiplekserów. Wejścia adresowe tych multiplekserów sterowane są przez specjalne rejestry, zawarte wewnątrz sterownika dynamicznej alokacji kanałów. Wartości zapisane w tych rejestrach tworzą więc konfigurację połączeń. Z uwagi na fakt, że konfiguracja ta może ulegać wielokrotnym zmianom podczas przebiegu procedury testowej, wektory testowe muszą zostać uzupełnione o dodatkowe informacje. Możliwości dynamicznej zmiany konfiguracji są następujące:

- uzupełnienie każdego wektora testowego o bity konfiguracyjne. Bity te mogą znajdować się na początku lub na końcu każdego wektora,
- wprowadzenie pojęcia wektorów konfiguracyjnych i związana z tym konieczność uzupełnienia każdego wektora o dodatkowy bit, którego wartość decyduje o tym, czy wektor ten będzie zawierać dane dla rejestrów dynamicznej alokacji kanałów, czy też dane testowe;

- wykorzystanie dodatkowego rejestru, o strukturze podobnej do budowy ścieżki testującej, dla którego przewidziany jest odrębny kanał testera lub który stanowi część interfejsu TAP (ang. *Test Access Port*).

Każde z wymienionych wyżej rozwiązań posiada zalety i wady zależne od wielkości SoC, układu połączeń zawartych w nim bloków EDT, oraz od możliwości zakodowania wektorów testowych dla każdego podukładu badanego. W przypadku pracy z EDT-SerDes prawdopodobnie nie będzie korzystne rezerwowanie jednego kanału na potrzeby konfiguracji. W rozdziale 6.3 niniejszej pracy podsumowano badanie zachowania EDT-SerDes pracującego z układem dynamicznej alokacji kanałów. Podczas eksperymentów wykorzystano sposób konfiguracji, wymagający wprowadzania wektorów konfiguracyjnych. Jest to rozwiązanie proste w implementacji i nie powodujące konieczności wprowadzania dużych zmian w zawartości wektorów testowych, wygenerowanych przez narzędzia Mentor Graphics, nie posiadających w chwili przeprowadzania prób zaimplementowanej technologii dynamicznej alokacji kanałów.



Rys. 4.1. Przykładowa struktura DUT w konfiguracji z dynamiczną alokacją kanałów

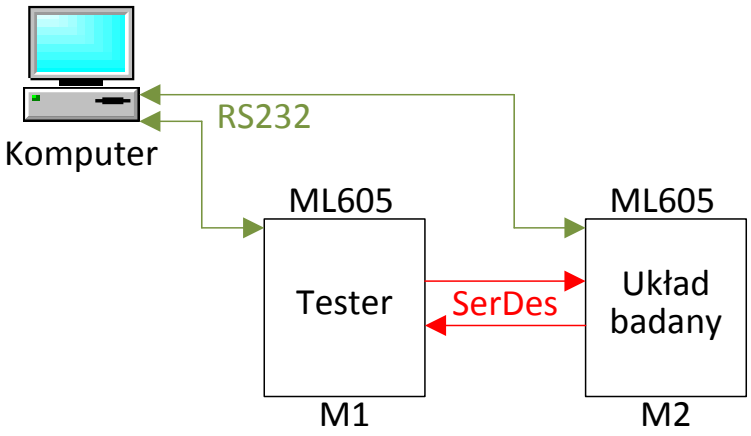
5. Implementacja w FPGA

5.1. Środowisko badawcze

Będące przedmiotem pracy łącze EDT-SerDes zostało zaimplementowane w strukturach układów FPGA Xilinx Virtex® 6, stanowiących główną jednostkę dwóch płyt ewaluacyjnych Xilinx ML605 [14]. Wspomniane układy FPGA posiadają sprzętowe moduły SerDes (Xilinx GTX) [15], umożliwiające przetestowanie funkcjonalności EDT-SerDes w niepełnym (ze względu na ograniczenia sprzętowe), ale dowodzącym pełni funkcjonalności łącza, zakresie.

Ogólna struktura systemu, wykorzystywanego do badań przedmiotu tej pracy, pokazana jest na rysunku 5.1. Widać na nim, że pracą całości steruje komputer PC, połączony za pomocą RS232 (emulowanego przez USB) z dwiema płytami ML605, z których jedna mimikuje tester, a druga – układ badany, obie płyty są zaś ze sobą połączone za pomocą szybkiego łącza SerDes (Xilinx GTX). Parametry widocznych na rysunku 5.1 połączeń są następujące:

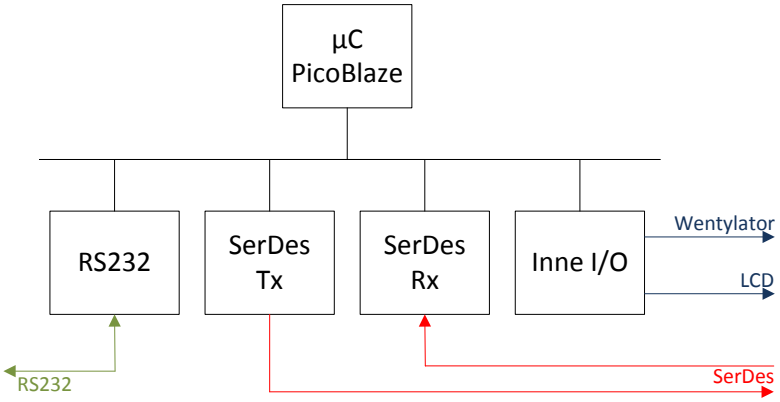
- RS232: szybkość 13840 bodów, efektywnie 11072 b/s,
- SerDes: 2 Gb/s, transmisja różnicowa w paśmie podstawowym, amplituda: 410mV.



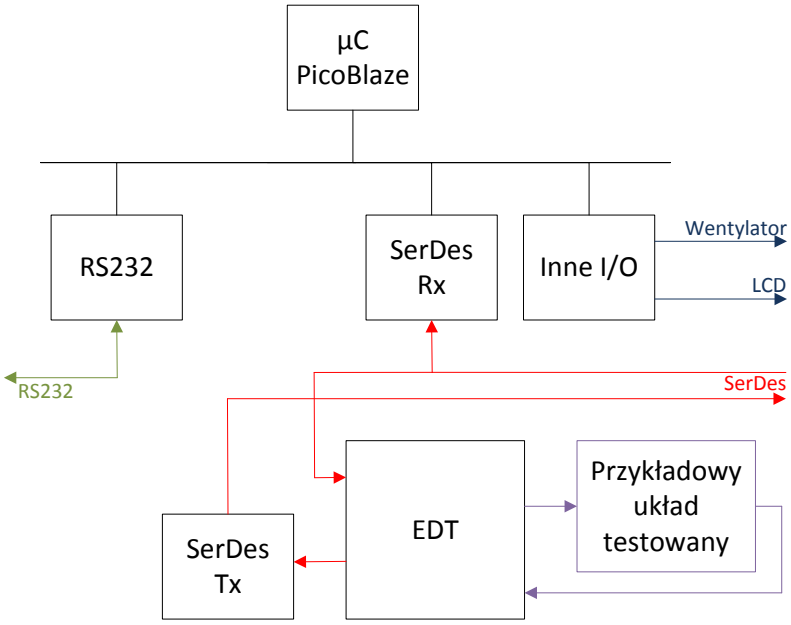
Rys. 5.1. Ogólna struktura środowiska badawczego

Wewnętrzna budowa obydwu modułów (M1 i M2 z rysunku 5.1) przedstawiona jest na rysunkach 5.2 i 5.3. Rolę nadrzędną w obu modułach pełni mikrokontroler PicoBlaze [16]. Ten prosty, 8-bitowy kontroler dostarczany jest przez firmę Xilinx w postaci zestawu modułów Verilog/VHDL i prostego asemblera. W jednym tylko przypadku PicoBlaze nie ma wpływu na działanie układu peryferyjnego – mowa tu o specyficznym połączeniu

nadajnika SerDes wewnątrz modułu M2. Nadajnik ten sterowany jest jedynie przez EDT, a ściślej – przesłanie danych wyzwalane jest poprzez odbiornik SerDes każdorazowo po odebraniu ramki transmisyjnej. Trzeba pamiętać, że choć nie wynika to z rysunków, sporządzonych celem zobrazowania struktury logicznej, nadajnik i odbiornik magistrali SerDes tworzą wspólną strukturę fizyczną.



Rys. 5.2. Schemat blokowy modułu (M1) emulującego tester



Rys. 5.3. Schemat blokowy układu (M2) emulującego układ badany z EDT

Ponadto, w rzeczywistości moduły M1 i M2 posiadają wspólny plik implementacyjny, a odpowiednim przekierowaniem niektórych magistrali wewnętrznych zarządza komputer PC. Innymi słowy oznacza to, że zsyntezowany kod opisu sprzętu jest wspólny dla obu

modułów, a komputer decyduje, która płytka będzie w danym momencie pełnić rolę testera, a która układu badanego. Takie rozwiązanie podyktowane jest chęcią oszczędzenia czasu, potrzebnego na syntezę projektu przez środowisko Xilinx ISE – czas potrzebny na wygenerowanie pliku programującego FPGA wynosi w przypadku tego projektu około 15 minut. Synteza dwóch oddzielnych projektów byłaby więc dużo bardziej czasochłonna.

5.2. Łącze EDT-SerDes

Sprzętowa warstwa łącza EDT-SerDes została zaimplementowana przy wykorzystaniu wbudowanych w układ FPGA Virtex 6 sprzętowych nadajników/odbiorników GTX. Każdy układ GTX charakteryzuje się teoretyczną możliwością pracy przy szybkości modulacji od 0,5 do 7,5 Gb/s. Moduły te przewidziane są do realizacji szybkich łączy szeregowych ogólnego użytku, z protokołami magistrali PCI, czy Serial-ATA, wykorzystywanych we współczesnych komputerach PC. Warto w tym miejscu zaznaczyć, że z racji przeznaczenia łączy GTX posiadają one wiele ograniczeń, z punktu widzenia EDT-SerDes. Pierwszym i największym zarazem jest fakt, że z poziomu logiki programowalnej FPGA nie ma bezpośredniego dostępu do magistrali szeregowej. Eliminuje to możliwość wykonywania operacji sekwencyjnych „bit-po-bicie" na transmitowanych danych i wymusza znalezienie rozwiązania alternatywnego. Najmniejszą dostępną dla FPGA porcją danych jest bajt, dlatego operacje sekwencyjne musiały zostać zastąpione operacjami kombinacyjnymi. Na przykład, zamiast stosowania układu rejestru liniowego służącego do wyznaczenia sumy kontrolnej, zgodnie z określonym wielomianem, musiały zostać wykorzystane komplementarne układy kombinacyjne z bramkami ExOR. Opis implementacji EDT-SerDes w FPGA, w języku Verilog, został przedstawiony w całości w dodatku A.

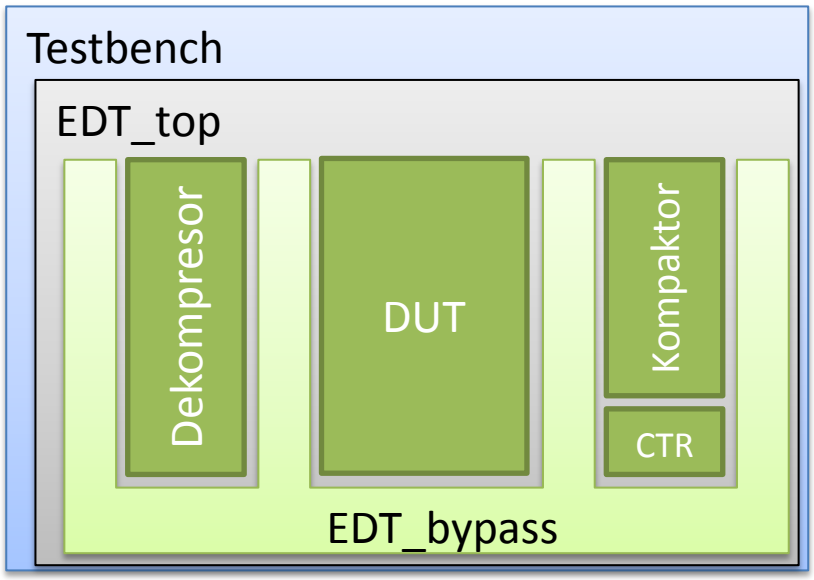
Wszystkie struktury implementacyjne i bloki funkcjonalne taktowane są sygnałem o częstotliwości 200 MHz. Do modułu EDT-SerDes doprowadzony jest sygnał 134 MHz, który zostaje zwielokrotniony w pętli fazowej do częstotliwości 2 GHz. Ten wysokoczęstotliwościowy sygnał taktuje sprzętowy nadajnik GTX, jest także transmitowany przez dzielnik częstotliwości ÷32 tak, by uzyskać sygnał 62,5 MHz. Ten ostatni sygnał taktuje wszystkie układy warstwy logicznej nadajnika/odbiornika SerDes (jest wykorzystany jako sygnał sterujący w głównym bloku *always@*). Ponieważ ramka transmisyjna składa się ze 128 bitów, maksymalna możliwa częstotliwość transmisji wynosi 15,6 MHz. Z taką

częstotliwością transmitowanych jest 100 kanałów, stąd wniosek, że efektywna szybkość transmisji za pomocą zaimplementowanego łącza EDT-SerDes wynosi 1,56 Gb/s. Jako że zaimplementowane łącze pracuje w trybie *full-duplex* (niezależna transmisja dwukierunkowa), podana wartość dotyczy tylko transmisji w jednym kierunku.

Układy GTX, jak wykazały doświadczenia, charakteryzują się stałym opóźnieniem, innym dla każdego układu. Opóźnienie to, choć niekorzystne dla transmisji „z oczekiwaniem na odpowiedź", pozwoliło bardzo udoskonalić protokół EDT-SerDes, który stał się odporny na błędy w obu kierunkach, przy jednoczesnym braku zwiększania zapotrzebowania na pojemność kanału transmisyjnego. Transmitery GTX, z uwagi na swoją konstrukcję i niemożność ingerowania w strumień binarny, wykluczyły także możliwość pełnego przetestowania sprawności łącza, z uwagi na fakt, że wewnątrz logicznej struktury nadajnika/odbiornika znajduje się mały układ sekwencyjny, pracujący z zegarem o częstotliwości zawsze równej 1/8 częstotliwości bitowej łącza. Jeśli wziąć pod uwagę, że maksymalna częstotliwość, przy jakiej producent FPGA gwarantuje poprawne działanie przerzutników, wynosi w tym przypadku 200 MHz łatwo zauważyć, że – teoretycznie – maksymalna bezpieczna częstotliwość pracy łącza wynosi 1,6 Gb/s. Podczas pracy przy 2 Gb/s wspomniany układ jest więc przetaktowany (ang. *overclocked*). Jest to niekorzystne zjawisko. Dalsze zwiększanie przepływności SerDes powodowało w testowanym module powstawanie błędów, związanych ze złą pracą przetaktowanego bloku. Mimo tego ograniczenia, zaimplementowane łącze można uznać za w pełni sprawny układ, którego funkcjonalność odpowiada podanym wcześniej wymaganiom i założeniom projektowym.

5.3. Symulacja EDT

EDT [11] (ang. *Embedded Deterministic Test*) jest nie tylko nazwą procedury przeprowadzania testu układów scalonych. Mianem EDT określa się również moduł otaczający poddawaną testowi strukturę wraz ze sterownikiem testu, zaimplementowany na tym samym podłożu co testowany układ i będący jego integralną częścią. Hierarchiczną strukturę modułów związanych z EDT przedstawia rysunek 5.4.



Rys. 5.4. Hierarchiczna struktura modułów technologii EDT: **Testbench** - struktura całości, uwzględniająca połączony z układem tester; **EDT_top** - moduł EDT wraz z całą wewnętrzną strukturą układu i blokami sterującymi testem; **Dekompresor** - dekompresor danych testowych; **DUT** – główna struktura układu scalonego (układ funkcjonalny), podlegająca testowi; **EDT_bypass** – blok, którego zadaniem jest umożliwienie połączenia pomiędzy pozostałymi blokami; **Kompaktor** – kompaktor odpowiedzi testowych; **CTR** – moduł kontrolny

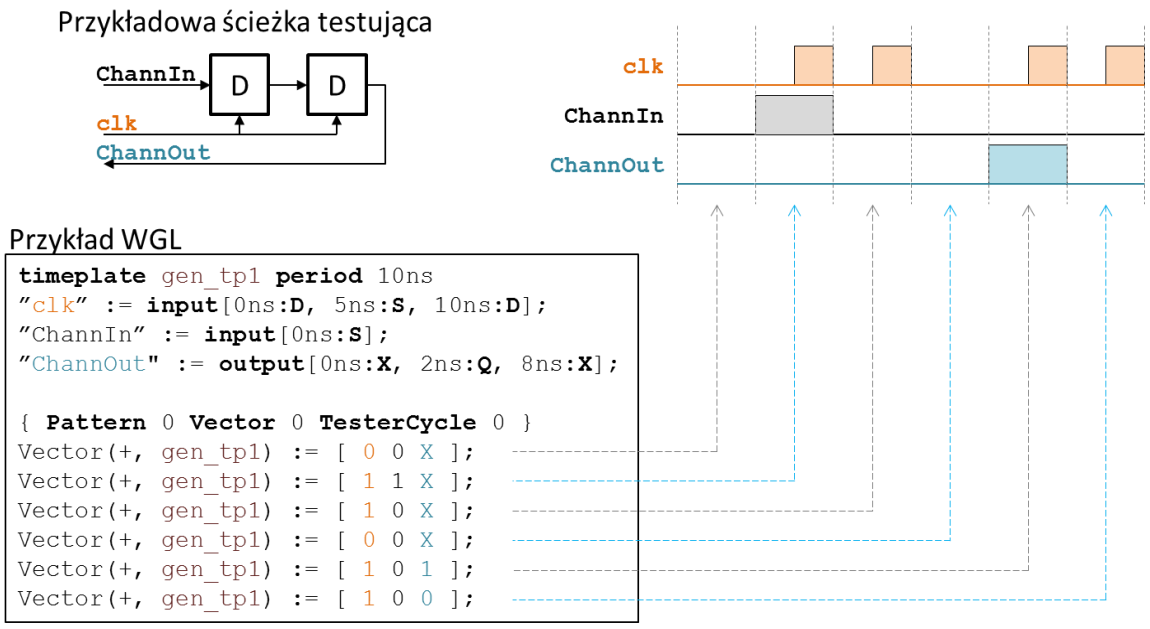
Wszystkie ukazane na rysunku 5.4 moduły musiały zostać zaimplementowane we wnętrzu tego modułu FPGA, którego zadaniem jest emulacja układu testowanego. Nie było jednak konieczne specjalne projektowanie ich pod kątem wykorzystania w FPGA, ponieważ narzędzia, wykonujące symulację uszkodzeń firmy Mentor Graphics jako efekt swego działania mogą zwracać opis całości w języku Verilog (EDT top wraz z modułami wewnętrznymi). Funkcjonalność ta została wykorzystana podczas realizacji opisywanego projektu. Jako układ badany wykorzystano prosty mikrokontroler PIC, o niewielkiej złożoności sprzętowej (w implementacji FPGA wykorzystuje 510 przerzutników i 489 tablic wartości funkcji LUT), pracujący w różnych konfiguracjach z punktu widzenia procedury

testowej. Zbadane w środowisku eksperymentalnym konfiguracje pracy kontrolera PIC zebrane są w tabeli 5.1.

Wspomniane wcześniej narzędzia wykonujące symulację uszkodzeń generują także zestaw wektorów testowych, koniecznych do wykonania testu badanego układu. Wektory te, uzupełnione o dane, służące do generacji sygnałów sterujących, zapisywane są w postaci plików, czytelnych dla oprogramowania zarządzającego testerami. Jednym z formatów tych plików, łatwym do odczytu i programowego parsowania, jest WGL. Przykładowy fragment takiego pliku wraz ze schematem prostego układu (ścieżki testującej – ang. *scan chain*) i wykresem przebiegów czasowych, został przedstawiony na rysunku 5.5. Widać na nim prostotę składni WGL i jest to uzasadnieniem wyboru tego formatu dla danych testowych na potrzeby przeprowadzania eksperymentów z EDT-SerDes.

Tabela 5.1. Badane konfiguracje pracy testowanego mikrokontrolera PIC

Konfiguracja	Cel eksperymentu
Jeden mikrokontroler PIC	Pokazanie, że EDT-SerDes jest transparentny dla EDT.
Jeden mikrokontroler PIC wraz ze zintegrowanym kontrolerem zegara (OCC – On-chip Clock Controller) firmy Mentor Graphics	Pokazanie, że EDT-SerDes wspiera technologię *At-Speed* przeprowadzania testów.
Dwa mikrokontrolery PIC, z których jeden wzbogacony jest o blok łączący wszystkie wejścia/wyjścia równoległe w rejestr szeregowy	Pokazanie, że EDT-SerDes może obsługiwać więcej niż jeden układ testowany i że każdy z badanych układów może pracować niezależnie od pozostałych.



Rys. 5.5. Przykład ilustrujący sposób zapisu danych w formacie WGL

Opisane wyżej dane testowe w formacie WGL wczytywane są przez program komputerowy, komunikujący się z płytami ewaluacyjnymi ML605, który dostosowuje dane do zaprogramowanego, zgodnie ze sprzętową implementacją, formatu. Operacja ta jest w istocie zamianą kolejności bitów w wierszach pliku WGL na przykład po to, by odpowiednio dostosować je do komunikacji z dwoma niezależnymi od siebie, lecz pracującymi równocześnie układami badanymi. Ponadto, program ten generuje wektor konfiguracyjny EDT-SerDes, służący do konfiguracji bloku bramkowania zegara po stronie odbiorczej łącza. Ten sam program odpowiedzialny jest także za przeprowadzenie procedury testowej. W tym celu przesyła kolejno wektory danych (wiersze pliku) do modułu M1 emulującego tester. M1, za pomocą EDT-SerDes przesyła każdy wektor do modułu M2, emulującego EDT. W odpowiedzi, M2 przesyła do M1 odpowiednią ramkę, zgodną z protokołem komunikacji. Dane z tej ramki przesyłane są przez M1 do komputera, który dokonuje analizy tych danych pod kątem poprawności pracy łącza i poprawnego przebiegu procedury testowej. Dodatkowo, oprogramowanie komputera umożliwia wprowadzanie błędów do ramek transmisyjnych, w celu testowania jakości protokołu EDT-SerDes. Wraz z dodatkową strukturą logiczną, zawartą na potrzeby eksperymentów w blokach EDT-SerDes, program PC umożliwia także przeprowadzanie dokładnych badań BER (ang. *bit error rate* – bitowa stopa błędów).

Pierwotnie całą procedurą testową miał zarządzać wyłącznie moduł M1, emulujący tester bez udziału komputera. Okazało się to jednak trudne, z uwagi na brak dostępnej w FPGA Virtex 6 pamięci RAM o rozmiarach, które umożliwiłyby przechowanie większej liczby wektorów testowych. Test z wykorzystaniem komputera nie wykorzystuje pełnej przepływności połączenia SerDes, należy jednak zauważyć, że każda ramka EDT-SerDes przesyłana jest z pełną szybkością (2 Gb/s), a pomiędzy ramkami przesyłane są bity, których zadaniem jest utrzymywanie pętli fazowej odbiornika w stałym synchronizmie z zegarem testera. Jest to ważna funkcjonalność EDT-SerDes. Taki tryb pracy nie jest związany wyłącznie z eksperymentami, stanowi bowiem uzasadnienie stwierdzenia, że EDT zawsze pracuje w synchronizmie z testerem niezależnie od tego, czy zegar kontrolera testu będzie pracować ze stałą, czy też zmienną częstotliwością. To z kolei jest jedną z konsekwencji założenia projektowego, jakim było zachowanie transparentności EDT-SerDes względem EDT.

Więcej niezgodności między projektem, a zaimplementowanym układem doświadczalnym, występuje w modułach synchronizacji i odzyskiwania sygnału czasu. Z uwagi na specyfikę bloków GTX, niemożliwe jest zgodne z projektem zaimplementowanie od-

biornika, pracującego przy EDT. W tym miejscu zaimplementowano układ synchronizacji identyczny z tym, który zawarty jest w bloku SerDes testera. Konsekwencją takiego stanu rzeczy może być zwiększone występowanie błędów i „gubienie” ramek transmisyjnych – sygnał taktujący odbiornik przy EDT może być bowiem niezsynchronizowany z sygnałem taktującym nadajnik testera. Jak pokazały eksperymenty, nie było to jednak przyczyną widocznej degradacji funkcjonalności badanego łącza EDT-SerDes.

6. Badania

6.1. Badania łącza

Pierwszym i najważniejszym eksperymentem, jakiemu poddano zaprojektowane łącze, było oszacowanie wartości BER (ang. *bit error rate* – bitowa stopa błędów) oraz sprawdzenie pracy łącza podczas emulowanego testu przykładowego układu scalonego, w różnych konfiguracjach.

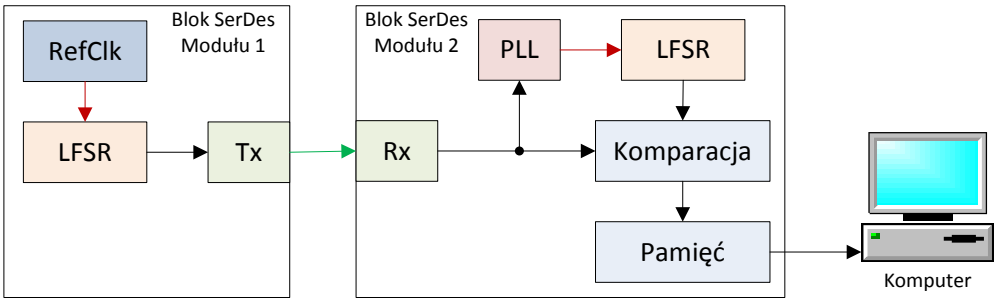
Z uwagi na fakt, że łącze GTX w jakie wyposażone są układy Xilinx Virtex 6 umożliwia ustalanie w pewnym zakresie parametrów elektrycznych, został sprawdzony wpływ na BER następujących wielkości:

- amplitudy (wartości napięcia) sygnału transmitowanego,
- kształtu charakterystyki częstotliwościowej wzmacniacza wejściowego odbiornika.

Warunki w jakich przeprowadzano badanie zostały zebrane w tabeli 6.1, a schemat blokowy układu do badania BER został przedstawiony na rysunku 6.1.

Tabela 6.1. Parametry pracy podczas badania BER

Parametr	Wartość			Jednostka
	Min	Stała	Max	
Amplituda sygnału	130	-	1750	mV
Szybkość bitowa	-	2	-	Gb/s
Długość ramki	-	128	-	b
Odstęp między ramkami	-	32	-	b
Długość przewodu koncentrycznego	-	30	-	cm



Rys. 6.1. Schemat blokowy układu do pomiaru BER

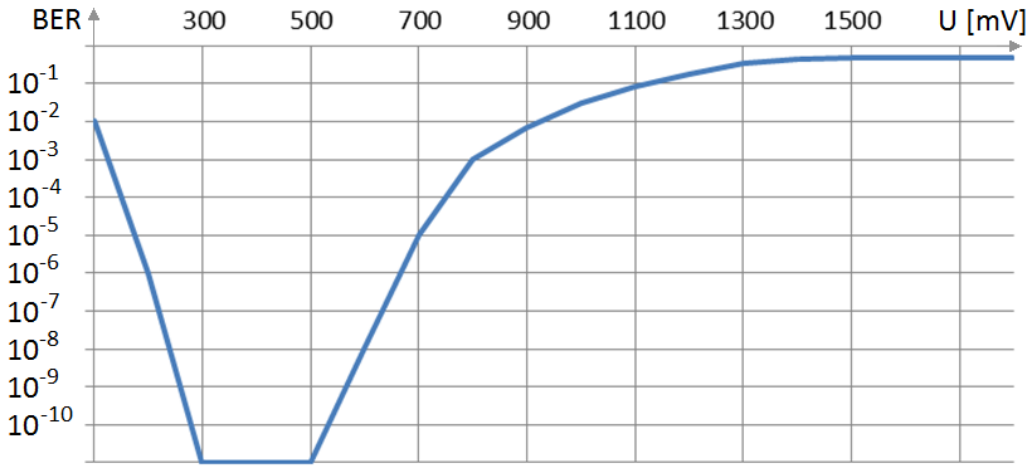
Jedna z płyt testowych pełni rolę nadajnika, druga – odbiornika i analizatora testowej sekwencji pseudo losowej. W obydwu modułach zaimplementowane są identyczne rejestry liniowe, inicjowane dokładnie taką samą wartością początkową. Na rysunku nie

oznaczono tego, lecz ważne jest, że sygnał inicjacji obydwu modułów wysyła komputer – najpierw do modułu odbierającego, potem do modułu nadającego sekwencję testową. Moduł odbierający oczekuje na nadejście pierwszej ramki testowej, wysłanej przez moduł transmitujący, tuż po jego inicjacji. Od tej chwili w ściśle określonych chwilach moduł pierwszy wysyła kolejną ramkę testową, a układ odbiorczy sprawdza zawartość buforu odbiornika. W ten sposób możliwe jest stwierdzenie, czy określona ramka została odebrana błędnie, czy też nie została odebrana w ogóle („zagubienie ramki" – na skutek błędu w sekwencji fazującej). Układ odbiorczy oblicza i buforuje w wewnętrznej pamięci podstawowe charakterystyki, służące następnie do identyfikacji charakteru BER, między innymi:

- średnia wartość liczby błędnych bitów w ramce,
- histogram liczności grup błędnych bitów w ramce,
- średnia wartość numeru pozycji w ramce, w której pojawiają się grupy błędów.

Charakterystyki te, wraz z innymi wartościami, przydatnymi podczas analizy pracy łącza, przesyłane są następnie do komputera i wyświetlane w odpowiedni dla danej charakterystyki sposób, przez oprogramowanie sterujące zestawem testowym.

W tym miejscu należy zwrócić uwagę na fakt, że wzmacniacz wejściowy bloku GTX układu Virtex 6 optymalizowany jest do pracy przy częstotliwości 2,5 GHz, dlatego wszystkie możliwe do wybrania charakterystyki częstotliwościowe filtru wejściowego posiadają maksimum w okolicach tej właśnie częstotliwości. Ponieważ podczas badania łącze pracowało z częstotliwością 2 GHz, a więc zbliżoną do częstotliwości optymalnej, prawdopodobnie nie okaże się zaskakujące, że badania nie wykazały żadnego wpływu wyboru charakterystyki wzmacniacza na BER. Największą uwagę przykładano zatem do sprawdzenia zależności BER od amplitudy sygnału transmitowanego. Wyniki badania przedstawione są w postaci wykresu BER=f(U) na rysunku 6.2. Warto zwrócić uwagę na gwałtowny wzrost stopy błędu przy wzroście napięcia sygnału powyżej 500 mV. Taki stan rzeczy spowodowany jest tym, że zmiana amplitudy sygnału nie skutkuje zmianą czułości wzmacniacza wejściowego, przy wyższych wartościach napięcia dochodziło więc do przesterowania obwodów wejściowych. Najważniejszym i konsekwentnie przestrzeganym w dalszych eksperymentach zaleceniem, wynikającym z analizy tego wykresu jest ograniczenie zakresu amplitudy sygnału do przedziału dokładnie 290 ÷ 510 mV. Podczas tego eksperymentu sygnały układu kontroli poprawności transmisji były ignorowane.



Rys. 6.2. Wynik badania BER w zależności od amplitudy

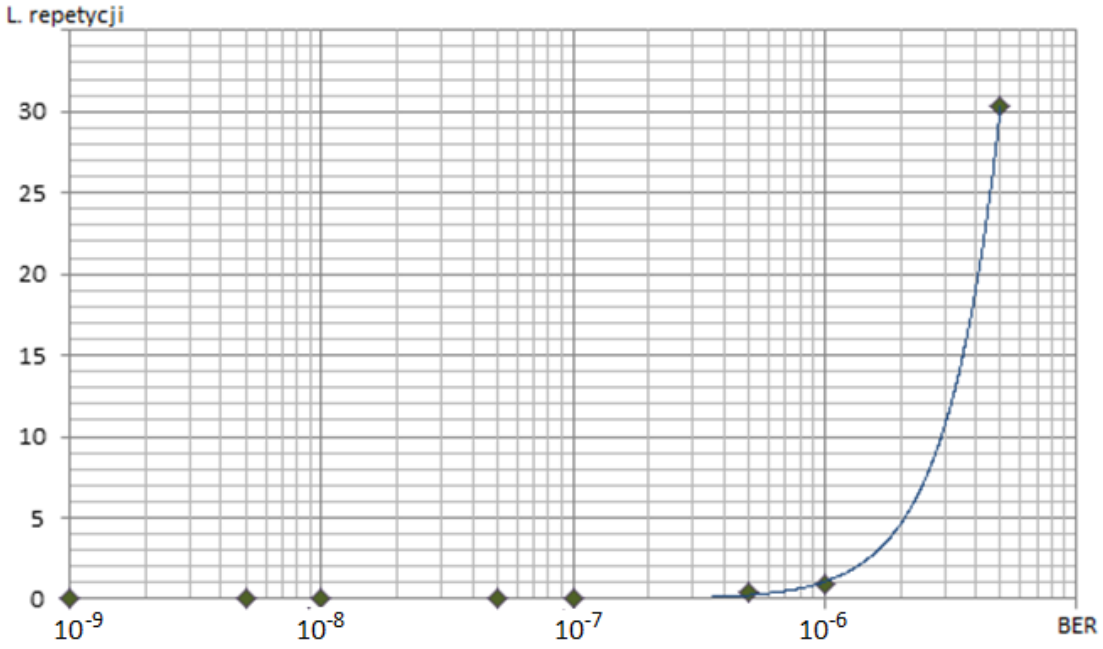
6.2. Emulacja i badanie procesu testowania

Łącze EDT-SerDes przeznaczone jest do transmisji danych charakterystycznej dla procesu testowania układów scalonych. Cechami takiej transmisji są między innymi: duża szybkość, wysoki determinizm czasowy, brak możliwości buforowania pełnego bloku danych (tworzącego pełen wektor testowy) przez odbiornik, a także niemożność oczekiwania przez tester na każdorazowe potwierdzanie poprawnego odbioru ramki, nim wysłana zostanie kolejna. Pracę łącza przetestowano więc badając zależność średniego czasu testowania oraz średniej liczby repetycji testu w zależności od BER. Każdorazowe wykrycie błędnej ramki podczas tego eksperymentu wiązało się z powtórzeniem całej procedury testowania. Amplituda sygnału podczas tego badania wynosiła około 420 mV, w celu zminimalizowania wartości bitowej stopy błędów, spowodowanej fizycznymi cechami układów Virtex 6. Następnie BER był „nastawiany" programowo, poprzez pseudo losowe wstrzykiwanie grup błędów, możliwie zgodnie z charakterystykami zebranymi podczas wcześniejszego badania bitowej stopy błędów. Tak więc liczność każdej grupy błędów wyznaczana była za pomocą generatora pseudo losowego o rozkładzie wykładniczym i wartości średniej 15, numer pierwszego błędnego bitu w ramce ustalany był przy użyciu generatora o rozkładzie równomiernym z zakresu 0 – 119 (ramka o długości 120

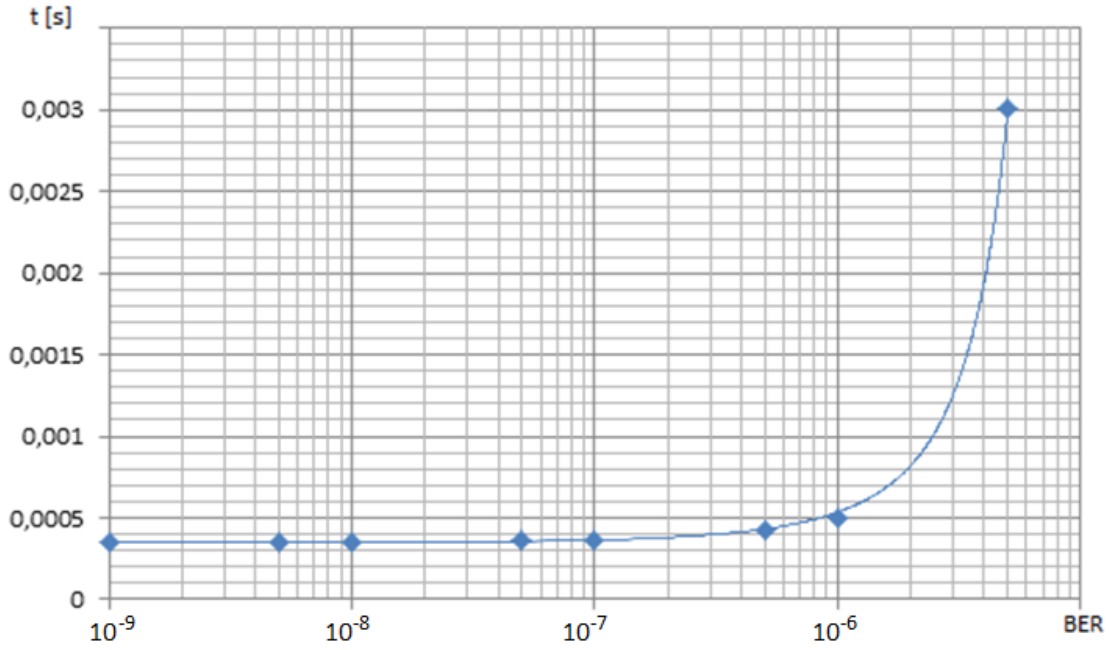
b), zaś numer ramki w której pojawi się błąd wyznaczany był z wykorzystaniem generatora o rozkładzie normalnym i parametrach wyliczanych zależnie od wybranej wartości BER.

W tym eksperymencie zaimplementowano w strukturze FPGA emulującego DUT przykładowy układ (mikrokontroler z rodziny PIC) uzupełniony przez narzędzie TestKompress firmy Mentor Graphics o ścieżki testujące (8 ścieżek, najdłuższa o długości 16 przerzutników) i kontroler testu – blok EDT. Zostało więc zaimplementowane pełne środowisko, w jakim docelowo ma pracować łącze EDT-SerDes. Podczas tego badania, oprócz zbierania wspomnianych wcześniej charakterystyk pracy łącza, sprawdzana była zgodność odpowiedzi DUT na aplikowane wektory testowe z wartościami oczekiwanymi. Ponieważ zaimplementowany układ nie posiadał uszkodzeń, każda niezgodność zostałaby zinterpretowana jako nieprawidłowość pracy jednego z bloków EDT-SerDes. Taka sytuacja się jednak nie zdarzyła.

Wyniki badania procedury testowej, na której przeprowadzenie potrzebnych było 5000 cykli zegarowych testera (co oznacza przesłanie 5000 ramek do DUT), przedstawione są w formie wykresów na rysunkach 6.3 i 6.4.



Rys. 6.3. Średnia liczba repetycji procedury testu w funkcji BER



Rys. 6.4. Średni czas testu DUT w funkcji BER

Pierwszy z wykresów obrazuje średnią liczbę repetycji. Warto zauważyć, że nawet przy BER rzędu 10^{-6} zdarzała się tylko jedna powtórka procedury, jednak dalszy wzrost stopy błędów w praktyce uniemożliwiał przeprowadzenie testu w rozsądnym czasie. Wykres kolejny przedstawia średni czas przeprowadzania procedury testowej. Na jego podstawie można stwierdzić, że BER na poziomie 10^{-6} zwiększa w widoczny sposób czas testu – to z kolei w widoczny sposób może przełożyć się na końcowy koszt produkcji. Niestety, nie jest możliwe jednoznaczne podanie recepty na zminimalizowanie czasu procedury testowej (uwzględniającej repetycje), ponieważ jest ona zależna od ilości koniecznych do przesłania danych, oraz od charakteru występowania błędów transmisji, który może być różny w zależności od fizycznych parametrów testera i połączenia go z DUT. Można jednak podać propozycję zminimalizowania czasu testu, niezależnie od liczby repetycji. Otóż podczas eksperymentu przyjmowano, że wykrycie błędu transmisji oznacza ponowne rozpoczęcie całej procedury testowej. Wiadomo jednak, że w rzeczywistości takie działanie nie musi być konieczne. Możliwe jest następujące rozwiązanie: wykrycie dowolnego błędu skutkuje ponownym przesłaniem uszkodzonego wektora testowego. Ponieważ jednak tester „nie wie", które słowo pamięci jest początkiem wektora, wszystkie słowa pamięci testera muszą zostać uzupełnione o jeden dodatkowy bit – znacznik początku wektora, lub „znacznik bezpiecznej repetycji". Wówczas, po wykryciu błędu transmisji, tester cofa wskaźnik adresowy do najbliższego słowa pamięci, którego wspomniany bit jest aktywowany. Tego rodzaju działanie znacznie skróci

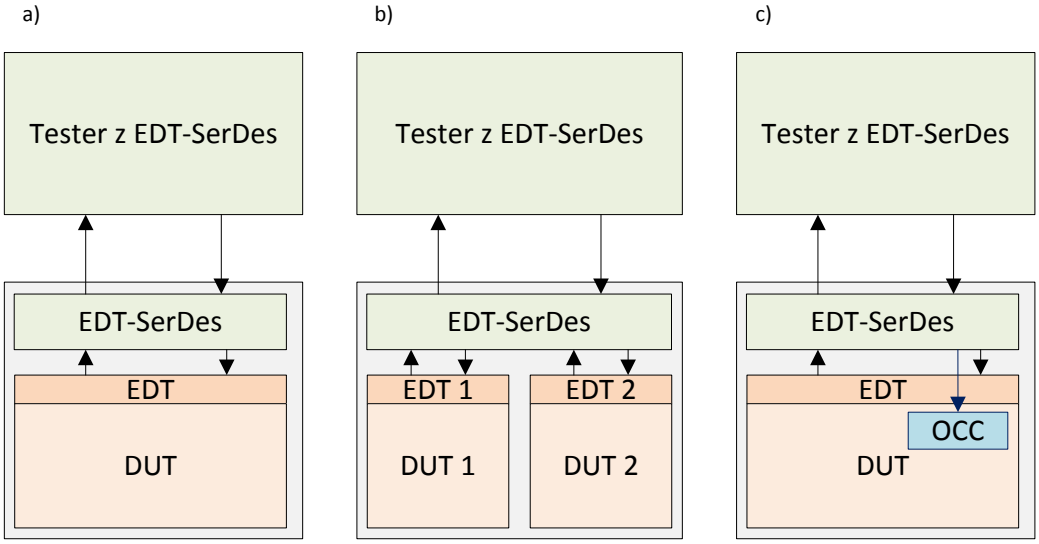
czas przeprowadzania testu niezależnie od wielkości BER, wymaga jednak uzupełnienia narzędzi generujących testy oraz testerów przeznaczonych do pracy z EDT-SerDes o obsługę takiej funkcjonalności. Zdaniem autora pracy jest to stosunkowo prosty i bardzo skuteczny zabieg.

6.3. Badanie funkcjonalności EDT-SerDes

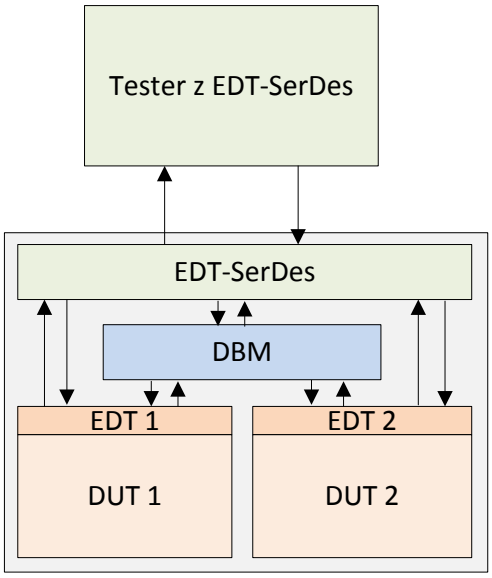
W końcowym etapie prac zaimplementowano i przedstawiono w działaniu różne konfiguracje DUT, współpracujące z łączem EDT-SerDes w celu wykazania pełnej zgodności proponowanego protokołu z popularnymi obecnie rozwiązaniami w dziedzinie testowania układów scalonych wielkiej skali integracji.

Sprawdzono pracę następujących konfiguracji DUT:

- jeden układ testowany (funkcjonalność podstawowa – rys. 6.5 a),
- dwa układy testowane za pomocą jednego łącza. Każdy z układów zawierał różną liczbę ścieżek testujących o różnych długościach (funkcjonalność podstawowa – rys. 6.5 b),
- jeden układ badany, z wykorzystaniem kontrolera zegara (OCC) (funkcjonalność At-speed – rys 6.5 c),
- dwa DUT badane z wykorzystaniem bloku dynamicznej alokacji kanałów (rys. 6.6, opis w rozdziale 4).



Rys. 6.5. Testowane konfiguracje pracy DUT i EDT-SerDes. a - funkcjonalność podstawowa, b - funkcjonalność podstawowa z dwoma różnymi DUT, c - funkcjonalność At-speed



Rys. 6.6. Testowana konfiguracja pracy EDT-SerDes z dynamiczną alokacją kanałów; DBM - ang. *dynamic bandwidth management* - blok dynamicznego zarządzania alokacją kanałów

Potwierdzono poprawność pracy i zgodność łącza z następującymi funkcjonalnościami:

- praca równoległa dowolnej liczby DUT z jednym łączem EDT-SerDes. Jedynym ograniczeniem jest długość ramki SerDes – jest to jednak wartość podlegająca konfiguracji. DUT nie muszą (choć mogą) ze sobą współpracować, każdy DUT może być testowany z wykorzystaniem oddzielnej procedury testowej;
- współpraca z OCC i generacja stabilnego, synchronicznego ze wszystkimi blokami układu sygnału taktującego dla pętli fazowej, zawartej we wspomnianym OCC, a więc zgodność z technologią *At-speed* firmy Mentor Graphics;
- współpraca z blokiem dynamicznej alokacji kanałów (według metody opisanej w rozdziale 4).

7. Podsumowanie

W pracy przedstawiono propozycję kompletnego łącza szeregowego, umożliwiającego współpracę testera z układem poddawanym testowi produkcyjnemu, przy jak najmniejszej ingerencji w strukturę badanego układu. Proponowane łącze charakteryzuje się transparentnością dla logiki EDT, wspiera wiele obecnie stosowanych technologii i narzędzi zaprojektowanych dla potrzeb testowania. Wykorzystuje szeroko obecnie stosowane nadajniki/odbiorniki SerDes. EDT-SerDes wymaga jedynie przekonstruowania testerów, ponieważ te stosowane obecnie wyposażone są jedynie w łącza cyfrowe, mogące pracować ze stosunkowo niską częstotliwością. Ponadto od testera EDT-SerDes wymaga się wyraźnego podziału bloku pamięci na pamięć wymuszeń i odpowiedzi, a także możliwości pracy bloku porównującego odpowiedzi DUT z dowolnym, niezależnym od testera opóźnieniem, względem bloku wysyłającego dane testowe.

EDT-SerDes w proponowanej postaci umożliwia niemal dowolną konfigurację liczby emulowanych kanałów równoległych, szybkość transmisji na łączu (co przekłada się na częstotliwość testu), oraz regulację podstawowych parametrów elektrycznych po stronie testera, dzięki czemu możliwe jest wykorzystanie istniejących, a nie umieszczanie dodatkowych układów szybkiej transmisji SerDes na powierzchni układu badanego. Temat pracy został zrealizowany, wykazano poprawność funkcjonowania zaproponowanego rozwiązania, możliwe jest przygotowanie EDT-SerDes w takiej postaci do implementacji i użytkowania podczas testu produkcyjnego. Należy jednak podkreślić, że EDT-SerDes w obecnej formie nie nadaje się do prowadzenia transmisji o charakterze innym niż podczas testowania układów scalonych i nie należy próbować stosować tego łącza w telekomunikacji, czy układach powszechnego użytku z powodu faktu, że przebieg i poprawność transmisji kontroluje zawsze tylko jedna ze stron i nie ma możliwości buforowania danych tak, by możliwa była realizacja komunikacji „z odpowiedzią”.

Prezentowane rozwiązanie jest bardzo elastyczne z punktu widzenia narzędzi generujących testy. Możliwa jest niemal dowolna manipulacja wielkością pola danych ramki transmisyjnej, co bezpośrednio przekłada się na liczbę emulowanych przez łącze szeregowe kanałów równoległych. Warto rozważyć także implementację EDT-SerDes z wykorzystaniem tylko jednego, dwukierunkowego połączenia fizycznego, co jednak może wiązać się z wydłużeniem czasu trwania procedury testowej. Autor pracy proponuje także możliwość korzystania z dwukierunkowego kanału transmisyjnego w taki sposób,

by porównywanie odpowiedzi DUT z wartościami oczekiwanymi następowało po stronie EDT, a stan komparacji przesyłany był do testera dopiero po zakończeniu transmisji całego wektora testowego. Sterownik EDT-SerDes po stronie DUT stałby się bardziej skomplikowany, pozwoliło by to jednak korzystać tylko z jednego fizycznego kanału SerDes przy nie wielkim tylko wydłużeniu procedury testowej. Prawdopodobnie wykorzystanie dwukierunkowej transmisji z wykorzystaniem jednego tylko kanału będzie kolejnym krokiem rozwoju EDT-SerDes.

8. Bibliografia

[1] System-Level Design Community. (2013, sierpień) [Online]. http://chipdesignmag.com/sld/blog/2013/07/25/experts-at-the-table-automotive-electronics/

[2] A. Athavale, C. Christensen, "High-speed serial I/O made simple. A designers' guide, with FPGA applications," Xilinx, 2005.

[3] H. Vranken, T. Vaajers, H. Fleury, D. Lelouver, "Enhanced reduced pin-count test for full-scan design," *International Test Conference, Proceedings*, 2001, str. 738 - 747.

[4] Atmel. (2013, maj) [Online]. http://www.atmel.com/Images/doc2503.pdf

[5] N. Mukherjee, W.-T. Cheng, S. Mahadevan, R. Press, J. Jahangiri, "Achieving high test quality with reduced pin count testing," *14th Asian Test Symposium, Proceedings*, 2005, str. 312 - 317.

[6] C. Papameletis et al., "SmartScan - reduced pin count compression with low power advantages ideal for mixed signal designs," *European Test Conference, Informal proceedings, artykuł 5C.3*, 2013.

[7] J. Moreau, T. Droniou, P. Lebourg, P. Armagnat, "Running scan test on three pins: yes, we can!," *International Test Conference, Proceedings*, 2009, str. 1 - 10.

[8] V. Tenentes, X. Kavousianos, "Test-data volume and scan power reduction with low ATE interface for multi-core SoCs," *ACM International Conference, Proceedings*, 2011, str. 747 - 754.

[9] L.-T. Wang, K. Abdel-Hafez, X. Wen, B. Sheu, S. Wu, S.-H. Lin, M.-T. Chang, "UltraScan: Using time-division demultiplexing/multiplexing (TDDM/TDM) with VirtualScan for test cost reduction," *International Test Conference, Proceedings, artykuł 36.4*, 2005.

[10] Xilinx Inc. (2013, maj) [Online]. http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf

[11] J. Rajski, J. Tyszer, M. Kassab, N. Mukherjee, "Embedded deterministic test," *IEEE Transactions on Computer-Aided Design*, nr 23, 2004, str. 776 - 792.

[12] M. Kassab, L. Xijiang, R. Morren, T. Waayers, "Clock control architecture and ATPG for reducing pattern count in SoC designs with multiple clock domains," *International Test Conference, Proceedings*, 2010.

[13] J. Janicki, M. Kassab, G. Mrugalski, N. Mukherjee, J. Rajski, J. Tyszer, "EDT bandwidth management in SoC designs," *IEEE Transactions on Computer-Aided Design*, nr 31, 2012, str. 1894 - 1907.

[14] Xilinx. (2013, maj) ML605 Evaluation Board. [Online]. http://www.xilinx.com/products/boards-and-kits/EK-V6-ML605-G.htm

[15] Xilinx Inc. (2013, sierpień) Xilinx - Virtex 6 GTX. [Online]. http://www.xilinx.com/support/documentation/user_guides/ug366.pdf

[16] Xilinx. (2013, maj) Xilinx PicoBlaze. [Online]. http://www.xilinx.com/products/intellectual-property/picoblaze.htm

Zawartość płyty CD:

- Wersja elektroniczna niniejszego dokumentu
- Projekt implementacji EDT-SerDes (Środowisko: Xilinx ISE 13)

9. Dodatki i uzupełnienia

Dodatek A. Implementacja łącza EDT-SerDes (Verilog)

```
// ****************************************************************
// MAIN EDT-SerDes MODULE
// ****************************************************************
module GTXWrapper3(
        input wire                  rx_n,           // Rx Negative
        input wire                  rx_p,           // Rx Positive
        input wire  [119:0]         d_in,           // Parallel in
        input wire  [7:0]           reset,          // [x,x,x,x,x,pll,tx,rx]
        input wire                  F6,             // RefClock Positive
        input wire                  F5,             // RefClock Negative
        input wire                  write,          // Write strobe
        output wire  [119:0]        d_out,          // Parallel out
        output wire                 pll_rx_kdet,    // PLL-OK
        output wire                 rx_reset_done,  // Reset-OK
        output wire                 tx_reset_done,  // Reset-OK
        output wire                 tx_n,           // Tx Negative
        output wire                 tx_p,           // Tx Positive
        input wire  [7:0]           diff_ctrl,      // Voltage control
        input wire  [7:0]           eq_mix,         // Amplifier control
        output wire                 gtx_clk,        // Output RefClock
        output wire                 strobe_out,     // Data received
        // EDT-SerDes specified
        output wire [119:0]         edt_bus, // Data bus valid [99:0] !!!
        output wire                 clk_out, // Clock for EDT
        input wire [7:0]            errors,  // To inject pseudo-random errors
        // Debug
        output wire [7:0]           debug,
        input wire                  prbs_reset,
        input wire                  prbs_enable,
        output wire [119:0]         prbs_err,
        output wire [119:0]         prbs_cnt,
        output wire [119:0]         test_port,
        input wire [7:0]            test_driver,
        input wire                  latch
    );

    wire                    tx_out_clk_int;
    wire        [1:0]       mgt_refclk_rx;
    reg         [31:0]      tx_data;
    wire        [31:0]      rx_data;
    reg         [159:0]     rx_register;
    reg         [127:0]     tx_register;
    wire        [7:0]       start_byte;
    wire        [31:0]      null_word;
    wire                    gtx_clk_inv;
    reg         [2:0]       rec_counter;
    wire                    rec_permission;
    wire                    gtx_rx_rst;
    wire                    gtx_tx_rst;
    wire                    pll_rx_rst;
    reg                     old_write;
    wire                    rx_rst;
    wire                    tx_rst;
```

```
reg                     strobe_out_int;
reg         [119:0]     d_out_int;
reg         [119:0]     prbs_errors;
reg         [3:0]       prbs_permission_cnt;
wire                    prbs_permission;
reg         [119:0]     prbs_cnt_int;
reg                     ber_run;
reg         [3:0]       lfsr_permission_cnt;
wire                    lfsr_permission;
wire        [119:0]     in_prbs;
wire        [119:0]     out_prbs;
reg                     prbs_o_n;
reg                     prbs_i_n;
wire        [119:0]     compare_err;
wire        [119:0]     compare_10;
reg         [119:0]     histo_1;
reg         [119:0]     histo_2;
reg         [119:0]     histo_3;
reg         [119:0]     histo_4;
wire        [119:0]     d_out_ber_10_mask;
reg         [119:0]     err_counter_10_01;
reg         [119:0]     old_drec;
reg         [119:0]     frames_lost;
wire        [119:0]     compare_01;
reg         [23:0]      frame_lost_serie_cnt;
reg         [119:0]     frame_lost_serie;
reg         [119:0]     test_port_int;
reg         [31:0]      last_position;
reg         [119:0]     bit_err;
wire                    tx_out_clk_int_nb;
wire        [7:0]       rx_data8;
wire        [7:0]       tx_data8;
wire                    tx_out_clk_d4;
wire                    tx_out_clk_d4_b;

// Debug (BER)
reg     [119:0]     frame1;
reg     [119:0]     lfsr1;
reg     [119:0]     frame2;
reg     [119:0]     lfsr2;
reg     [119:0]     frame3;
reg     [119:0]     lfsr3;
reg     [119:0]     frame4;
reg     [119:0]     lfsr4;
reg     [119:0]     frame5;
reg     [119:0]     lfsr5;
reg     [119:0]     frame6;
reg     [119:0]     lfsr6;
reg     [119:0]     frame7;
reg     [119:0]     lfsr7;
reg     [119:0]     frame8;
reg     [119:0]     lfsr8;
reg     [5:0]       flx;
reg     [119:0]     latched_data;
reg     [119:0]     latched_lfsr;
reg     [119:0]     latched_fnum;

wire    [119:0]     d_in_crc;
wire    [119:0]     d_in_with_flags;
wire                rx_crc_ok;
wire    [3:0]       rx_counter;
```

```
wire    [3:0]       tx_counter;
reg                 tx_counter_inc;
reg                 rx_counter_inc;
reg                 tx_counter_reset;
reg                 rx_counter_reset;
reg                 tx_counter_must_reset_1;
reg                 tx_counter_must_reset_2;
reg                 rx_counter_must_reset_1;
reg                 rx_counter_must_reset_2;
reg                 frame_lost;

wire   LFC; // 1 if last send frame was OK
assign d_in_with_flags [119]     = d_in[119];
assign d_in_with_flags [118]     = rx_crc_ok;
assign d_in_with_flags [117:114] = tx_counter [3:0];
assign d_in_with_flags [113]     = frame_lost;
assign d_in_with_flags [112:0]   = d_in[112:0];
assign LFC                       = d_out[118];

CRC8 TxCRC (
.DataIn(d_in_with_flags),
.DataOut(d_in_crc),
.CompareOK(),
 .error(errors[1])
);
CRC8 RxCRC (
.DataIn(d_out),
.DataOut(),
.CompareOK(rx_crc_ok),
 .error(1'b0)
);
frame_counter tx_counter_module (
.increment(tx_counter_inc),
.reset(tx_counter_reset),
.state(tx_counter)
);
frame_counter rx_counter_module (
.increment(rx_counter_inc),
.reset(rx_counter_reset),
.state(rx_counter)
);

    // EDT-SerDes
    wire            clk_mask_write;
    reg     [99:0]  clk_mask;
    reg     [99:0]  edt_bus_int;
    reg             edt_strobe_int_pre;
    reg             edt_strobe_int;

    assign clk_out = edt_strobe_int;
    assign clk_mask_write = d_out_int[119];
    assign edt_bus[99:0] = edt_bus_int & ({100{edt_strobe_int}} | (~clk_mask));

 OBUF dioda (
    .I(prbs_enable),
    .O(debug[0]) );
 OBUF dioda2 (
    .I(strobe_out),
    .O(debug[7]) );
 OBUF dioda3 (
    .I(prbs_permission),
```

```
    .O(debug[1]) );

assign debug[6:2]      = 5'b0;
assign test_port       = test_port_int;
assign prbs_cnt        = prbs_cnt_int;
assign prbs_err        = prbs_errors;
assign prbs_permission = (prbs_permission_cnt == 4'b0);
assign strobe_out      = strobe_out_int;
assign d_out           = d_out_int;
assign start_byte      = 8'b11001001;
assign null_word       = 32'b10011111101001111110100111111000;
assign rec_permission  = (rec_counter == 0);
assign pll_rx_rst      = reset[2];
assign gtx_tx_rst      = reset[1];
assign gtx_rx_rst      = reset[0];
assign rx_rst          = reset[3];
assign tx_rst          = reset[4];
assign gtx_clk_inv     = ~tx_out_clk_d4;
assign gtx_clk         = ~gtx_clk_inv;
assign lfsr_permission = (lfsr_permission_cnt == 0);


initial begin
    tx_data             <=  0;
    tx_register         <=  0;
    rx_register         <=  0;
    rec_counter         <=  0;
    strobe_out_int      <=  0;
    old_write           <=  0;
    d_out_int           <= 120'b0;
    prbs_errors         <= 120'b0;
    prbs_o_n            <= 1'b0;
    prbs_i_n            <= 1'b0;
    prbs_permission_cnt <= 4'b0;
    prbs_cnt_int        <= 120'b0;
    ber_run             <= 0;
    lfsr_permission_cnt <= 0;
    histo_1             <= 120'b0;
    histo_2             <= 120'b0;
    histo_3             <= 120'b0;
    histo_4             <= 120'b0;
    err_counter_10_01   <= 120'b0;
    old_drec            <= 120'b0;
    frames_lost         <= 120'b0;
    frame_lost_serie_cnt <= 24'b0;
    frame_lost_serie    <= 120'b0;
    test_port_int       <= 120'b0;
    last_position       <= 0;
    bit_err             <= 0;
    flx                 <= 0;
    latched_data        <= 0;
    latched_lfsr        <= 0;
    latched_fnum        <= 0;
    clk_mask            <= 0;
    edt_strobe_int      <= 0;
    edt_strobe_int_pre  <= 0;
    edt_bus_int         <= 0;
    tx_counter_reset    <= 0;
    rx_counter_reset    <= 0;
    tx_counter_inc      <= 0;
    rx_counter_inc      <= 0;
```

```
    frame_lost          <= 0;
    tx_counter_must_reset_1 <= 0;
    tx_counter_must_reset_2 <= 0;
    rx_counter_must_reset_1 <= 0;
    rx_counter_must_reset_2 <= 0;
end

//************************************
//              SerDes               *
//************************************

// To detect start-sequence:
//    00100000000100000... - sequence detection
wire [31:0] rec_detect_z;
//    00100000000000000... - first sequence selection
wire [31:0] rec_detect;
assign rec_detect_z[0] = (rx_register[159-0 : 152-0] == start_byte[7:0]);
assign rec_detect[0] = rec_detect_z[0];
genvar i;
generate
    for (i=1; i<32; i=i+1) begin
        assign rec_detect_z[i] =
               (rx_register[159-i : 152-i] == start_byte[7:0]);
        assign rec_detect[i] =
               rec_detect_z[i] & (rec_detect_z[i-1:0] == 0);
    end
endgenerate

// to measure BER
Compare120 prbs_errors_c (120'b0, d_out_ber_10_mask, compare_10);
Compare120 prbs_errors_01_10 (d_out_int, in_prbs, compare_err);
PRBS_generator2 in_prbs_m (tx_out_clk_d4_b, prbs_reset, prbs_i_n, in_prbs);
PRBS_generator2 out_prbs_m (tx_out_clk_d4_b, prbs_reset, prbs_o_n, out_prbs);
assign  d_out_ber_10_mask = (d_out_int ^ in_prbs) || in_prbs;
assign  compare_01 = compare_err - compare_10;

// === MAIN LOOP - Tx, Rx, BER, EDT =======
always @ (posedge tx_out_clk_d4_b) begin
    // --- Test Port - for debug only --------
    if (test_driver == 0) begin
    end else if (test_driver == 1) begin
        test_port_int <= histo_2;
    end else if (test_driver == 2) begin
        test_port_int <= histo_3;
    end else if (test_driver == 3) begin
        test_port_int <= histo_4;
    end else if (test_driver == 4) begin
        test_port_int <= err_counter_10_01;
    end else if (test_driver == 5) begin
        test_port_int <= frames_lost;
    end else if (test_driver == 6) begin
        test_port_int <= frame_lost_serie;
    end else if (test_driver == 7) begin
        test_port_int <= frame1;
    end else if (test_driver == 8) begin
        test_port_int <= lfsr1;
    end else if (test_driver == 9) begin
        test_port_int <= frame2;
    end else if (test_driver == 10) begin
        test_port_int <= lfsr2;
    end else if (test_driver == 11) begin
```

```
        test_port_int <= bit_err;
    end else if (test_driver == 12) begin
        test_port_int <= frame3;
    end else if (test_driver == 13) begin
        test_port_int <= lfsr3;
    end else if (test_driver == 14) begin
        test_port_int <= frame4;
    end else if (test_driver == 15) begin
        test_port_int <= lfsr4;
    end else if (test_driver == 16) begin
        test_port_int <= frame5;
    end else if (test_driver == 17) begin
        test_port_int <= lfsr5;
    end else if (test_driver == 18) begin
        test_port_int <= latched_data;
    end else if (test_driver == 19) begin
        test_port_int <= latched_lfsr;
    end else if (test_driver == 20) begin
        test_port_int <= latched_fnum;
    end else if (test_driver == 21) begin
        test_port_int <= frame6;
    end else if (test_driver == 22) begin
        test_port_int <= lfsr6;
    end else if (test_driver == 23) begin
        test_port_int <= frame7;
    end else if (test_driver == 24) begin
        test_port_int <= lfsr7;
    end else if (test_driver == 25) begin
        test_port_int <= frame8;
    end else if (test_driver == 26) begin
        test_port_int <= lfsr8;
    end else begin
        test_port_int <= 120'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF;
    end
    // --- PRBS / BER --------------------
    if (prbs_reset) begin
        prbs_errors <= 120'b0;
        prbs_i_n <= 1'b0;
        prbs_o_n <= 1'b0;
        prbs_cnt_int <= 120'b0;
        ber_run <= 0;
        lfsr_permission_cnt <= 0;
        histo_1 <= 120'b0;
        histo_2 <= 120'b0;
        histo_3 <= 120'b0;
        histo_4 <= 120'b0;
        err_counter_10_01 <= 120'b0;
        old_drec <= 120'b0;
        frames_lost <= 120'b0;
        frame_lost_serie_cnt <= 24'b0;
        frame_lost_serie <= 120'b0;
        bit_err <= 0;
        flx <= 0;
    end else if (prbs_enable) begin
        if (ber_run) begin
            if (~lfsr_permission) begin
                lfsr_permission_cnt <= lfsr_permission_cnt - 1;
                prbs_i_n <= 1'b0;
            end else begin
                old_drec <= d_out_int;
                lfsr_permission_cnt <= 4;
```

```
prbs_i_n <= 1'b1;
//============== BER ==================
prbs_cnt_int <= prbs_cnt_int + 1;    // total FrameCounter
if (old_drec == d_out_int) begin
    // frame lost
    frames_lost <= frames_lost + 1;
    frame_lost_serie_cnt <= frame_lost_serie_cnt + 1;
end else begin
    if (latch) begin
        latched_data <= d_out_int;
        latched_lfsr <= in_prbs;
        latched_fnum <= prbs_cnt_int + 1;
    end
    if (flx == 0) begin
        flx <= 1;
        frame2 <= d_out_int;
        lfsr2 <= in_prbs;
    end else if (flx == 1) begin
        flx <= 2;
        frame3 <= d_out_int;
        lfsr3 <= in_prbs;
    end else if (flx == 2) begin
        flx <= 3;
        frame4 <= d_out_int;
        lfsr4 <= in_prbs;
    end else if (flx == 3) begin
        flx <= 4;
        frame5 <= d_out_int;
        lfsr5 <= in_prbs;
    end else if (flx == 4) begin
        flx <= 5;
        frame6 <= d_out_int;
        lfsr6 <= in_prbs;
    end else if (flx == 5) begin
        flx <= 6;
        frame7 <= d_out_int;
        lfsr7 <= in_prbs;
    end else if (flx == 6) begin
        flx <= 7;
        frame8 <= d_out_int;
        lfsr8 <= in_prbs;
    end
    bit_err <= bit_err + compare_err;       // BITS
    if (d_out_int != in_prbs) begin         // FRAMES
        prbs_errors <= prbs_errors + 1;
    end
    //--- histogram ---
    if (compare_err > 0 && compare_err <= 10) begin
        histo_1[119:80] <= histo_1[119:80] + 1;
    end else if (compare_err > 0 && compare_err <= 20) begin
        histo_1[79:40] <= histo_1[79:40] + 1;
    end else if (compare_err > 0 && compare_err <= 30) begin
        histo_1[39:0] <= histo_1[39:0] + 1;
    end else if (compare_err > 0 && compare_err <= 40) begin
        histo_2[119:80] <= histo_2[119:80] + 1;
    end else if (compare_err > 0 && compare_err <= 50) begin
        histo_2[79:40] <= histo_2[79:40] + 1;
    end else if (compare_err > 0 && compare_err <= 60) begin
        histo_2[39:0] <= histo_2[39:0] + 1;
    end else if (compare_err > 0 && compare_err <= 70) begin
        histo_3[119:80] <= histo_3[119:80] + 1;
```

```
                end else if (compare_err > 0 && compare_err <= 80) begin
                    histo_3[79:40] <= histo_3[79:40] + 1;
                end else if (compare_err > 0 && compare_err <= 90) begin
                    histo_3[39:0] <= histo_3[39:0] + 1;
                end else if (compare_err > 0 && compare_err <= 100) begin
                    histo_4[119:80] <= histo_4[119:80] + 1;
                end else if (compare_err > 0 && compare_err <= 110) begin
                    histo_4[79:40] <= histo_4[79:40] + 1;
                end else if (compare_err > 0 && compare_err <= 120) begin
                    histo_4[39:0] <= histo_4[39:0] + 1;
                end
                //-- 0->1 1->0 ----
                err_counter_10_01 [119:60] <= err_counter_10_01 [119:60] +
                  compare_10 [59:0];
                err_counter_10_01 [59:0] <= err_counter_10_01 [59:0] +
                  compare_01 [59:0];
                //-- frame lost serie -
                if (frame_lost_serie_cnt == 1) begin
                 if (frame_lost_serie [119:96] != 24'hFFFFFF)
                frame_lost_serie [119:96] <= frame_lost_serie [119:96] + 1;
                 end else if (frame_lost_serie_cnt == 2) begin
                if (frame_lost_serie [95:72] != 24'hFFFFFF)
                  frame_lost_serie [95:72] <= frame_lost_serie [95:72] + 1;
                end else if (frame_lost_serie_cnt == 3) begin
                if (frame_lost_serie [71:48] != 24'hFFFFFF)
                  frame_lost_serie [71:48] <= frame_lost_serie [71:48] + 1;
                end else if (frame_lost_serie_cnt == 4) begin
                if (frame_lost_serie [47:24] != 24'hFFFFFF)
                  frame_lost_serie [47:24] <= frame_lost_serie [47:24] + 1;
                end else if (frame_lost_serie_cnt != 0) begin
                if (frame_lost_serie [23:0] != 24'hFFFFFF)
                  frame_lost_serie [23:0] <= frame_lost_serie [23:0] + 1;
                end
                frame_lost_serie_cnt <= 0;
            end
        end
    end else if (strobe_out_int) begin
        if (~ber_run) begin
            ber_run <= 1;
            prbs_i_n <= 1'b1;
            lfsr_permission_cnt <= 5;
            frame1 <= d_out_int;
            lfsr1 <= in_prbs;
        end
    end else begin
        prbs_i_n <= 1'b0;
    end
end
// --- send new data OR regular transmission
tx_data                <=  tx_register [127:96];
if (~old_write & write | prbs_permission & prbs_enable & ~prbs_reset) begin
    if (prbs_enable) begin
        tx_register [119:0] <=  out_prbs;
        prbs_o_n <= 1'b1;
        prbs_permission_cnt <= 4;
        tx_counter_inc <= 0;
        tx_counter_must_reset_1 <= 0;
    end else begin
        tx_register [119:0] <=  d_in_crc;
        if (d_in[119] == 1'b0) begin                    // Data frame
            tx_counter_inc <= 1;
```

```
            tx_counter_must_reset_1 <= 0;
            rx_counter_must_reset_2 <= 0;
        end else begin                                  // Config frame
            tx_counter_inc <= 0;
            tx_counter_must_reset_1 <= 1;
            rx_counter_must_reset_2 <= 1;
        end
    end
    tx_register [127:120]   <=  start_byte;
end else begin
    tx_counter_inc <= 0;
    tx_counter_must_reset_1 <= 0;
    rx_counter_must_reset_2 <= 0;
    tx_register [127:32]    <=  tx_register [95:0];
    tx_register [31:0]  <= null_word;
    prbs_o_n <= 1'b0;
    if (~prbs_permission & ~prbs_reset) begin
        prbs_permission_cnt <= prbs_permission_cnt - 1;
    end
end
// check for new received data
    if (rec_permission) begin
        case (rec_detect[31:0])
        1: begin
            d_out_int           <= rx_register [159-8-0 : 32-0];
            rec_counter <= 4;
            strobe_out_int  <= 1;
            last_position <= rec_detect;
        end
        2: begin
            d_out_int           <= rx_register [159-8-1 : 32-1];
            rec_counter <= 4;
            strobe_out_int  <= 1;
            last_position <= rec_detect;
        end
        4: begin
            d_out_int           <= rx_register [159-8-2 : 32-2];
            rec_counter <= 4;
            strobe_out_int  <= 1;
            last_position <= rec_detect;
        end
        8: begin
            d_out_int           <= rx_register [159-8-3 : 32-3];
            rec_counter <= 4;
            strobe_out_int  <= 1;
            last_position <= rec_detect;
        end
        16: begin
            d_out_int           <= rx_register [159-8-4 : 32-4];
            rec_counter <= 4;
            strobe_out_int  <= 1;
            last_position <= rec_detect;
        end
        32: begin
            d_out_int           <= rx_register [159-8-5 : 32-5];
            rec_counter <= 4;
            strobe_out_int  <= 1;
            last_position <= rec_detect;
        end
        64: begin
            d_out_int           <= rx_register [159-8-6 : 32-6];
```

```
    rec_counter <= 4;
    strobe_out_int  <= 1;
    last_position <= rec_detect;
end
128: begin
    d_out_int           <= rx_register [159-8-7 : 32-7];
    rec_counter <= 4;
    strobe_out_int  <= 1;
    last_position <= rec_detect;
end
256: begin
    d_out_int           <= rx_register [159-8-8 : 32-8];
    rec_counter <= 4;
    strobe_out_int  <= 1;
    last_position <= rec_detect;
end
512: begin
    d_out_int           <= rx_register [159-8-9 : 32-9];
    rec_counter <= 4;
    strobe_out_int  <= 1;
    last_position <= rec_detect;
end
1024: begin
    d_out_int           <= rx_register [159-8-10 : 32-10];
    rec_counter <= 4;
    strobe_out_int  <= 1;
    last_position <= rec_detect;
end
2048: begin
    d_out_int           <= rx_register [159-8-11 : 32-11];
    rec_counter <= 4;
    strobe_out_int  <= 1;
    last_position <= rec_detect;
end
4096: begin
    d_out_int           <= rx_register [159-8-12 : 32-12];
    rec_counter <= 4;
    strobe_out_int  <= 1;
    last_position <= rec_detect;
end
8192: begin
    d_out_int           <= rx_register [159-8-13 : 32-13];
    rec_counter <= 4;
    strobe_out_int  <= 1;
    last_position <= rec_detect;
end
16384: begin
    d_out_int           <= rx_register [159-8-14 : 32-14];
    rec_counter <= 4;
    strobe_out_int  <= 1;
    last_position <= rec_detect;
end
32768: begin
    d_out_int           <= rx_register [159-8-15 : 32-15];
    rec_counter <= 4;
    strobe_out_int  <= 1;
    last_position <= rec_detect;
end
65536: begin
    d_out_int           <= rx_register [159-8-16 : 32-16];
    rec_counter <= 4;
```

```
    strobe_out_int  <= 1;
    last_position <= rec_detect;
end
131072: begin
    d_out_int           <= rx_register [159-8-17 : 32-17];
    rec_counter <= 4;
    strobe_out_int  <= 1;
    last_position <= rec_detect;
end
262144: begin
    d_out_int           <= rx_register [159-8-18 : 32-18];
    rec_counter <= 4;
    strobe_out_int  <= 1;
    last_position <= rec_detect;
end
524288: begin
    d_out_int           <= rx_register [159-8-19 : 32-19];
    rec_counter <= 4;
    strobe_out_int  <= 1;
    last_position <= rec_detect;
end
1048576: begin
    d_out_int           <= rx_register [159-8-20 : 32-20];
    rec_counter <= 4;
    strobe_out_int  <= 1;
    last_position <= rec_detect;
end
2097152: begin
    d_out_int           <= rx_register [159-8-21 : 32-21];
    rec_counter <= 4;
    strobe_out_int  <= 1;
    last_position <= rec_detect;
end
4194304: begin
    d_out_int           <= rx_register [159-8-22 : 32-22];
    rec_counter <= 4;
    strobe_out_int  <= 1;
    last_position <= rec_detect;
end
8388608: begin
    d_out_int           <= rx_register [159-8-23 : 32-23];
    rec_counter <= 4;
    strobe_out_int  <= 1;
    last_position <= rec_detect;
end
16777216: begin
    d_out_int           <= rx_register [159-8-24 : 32-24];
    rec_counter <= 4;
    strobe_out_int  <= 1;
    last_position <= rec_detect;
end
33554432: begin
    d_out_int           <= rx_register [159-8-25 : 32-25];
    rec_counter <= 4;
    strobe_out_int  <= 1;
    last_position <= rec_detect;
end
67108864: begin
    d_out_int           <= rx_register [159-8-26 : 32-26];
    rec_counter <= 4;
    strobe_out_int  <= 1;
```

```
            last_position <= rec_detect;
        end
        134217728: begin
            d_out_int           <= rx_register [159-8-27 : 32-27];
            rec_counter <= 4;
            strobe_out_int  <= 1;
            last_position <= rec_detect;
        end
        268435456: begin
            d_out_int           <= rx_register [159-8-28 : 32-28];
            rec_counter <= 4;
            strobe_out_int  <= 1;
            last_position <= rec_detect;
        end
        536870912: begin
            d_out_int           <= rx_register [159-8-29 : 32-29];
            rec_counter <= 4;
            strobe_out_int  <= 1;
            last_position <= rec_detect;
        end
        1073741824: begin
            d_out_int           <= rx_register [159-8-30 : 32-30];
            rec_counter <= 4;
            strobe_out_int  <= 1;
            last_position <= rec_detect;
        end
        2147483648: begin
            d_out_int           <= rx_register [159-8-31 : 32-31];
            rec_counter <= 4;
            strobe_out_int  <= 1;
            last_position <= rec_detect;
        end
        endcase
    end
    if (rec_detect[31:0] == 32'b0 | ~rec_permission) begin
        if (~rec_permission) begin
            rec_counter <= rec_counter - 1;
        end
        strobe_out_int  <= 0;
    end
    if (strobe_out_int) begin
        if (clk_mask_write) begin
            clk_mask [99:0] <= d_out_int[99:0];
            edt_strobe_int_pre <= 0;
            rx_counter_inc <= 0;
            rx_counter_must_reset_1 <= 1;
            tx_counter_must_reset_2 <= 1;
            frame_lost <= 0;
        end else begin
            edt_bus_int [99:0] <= d_out_int[99:0];
            edt_strobe_int_pre <= 1;
            if (d_out_int [117:114] == rx_counter [3:0]) begin
                frame_lost <= 0;
            end else begin
                frame_lost <= 1;
            end
            if (~errors[0]) rx_counter_inc <= 1;
            rx_counter_must_reset_1 <= 0;
            tx_counter_must_reset_2 <= 0;
        end
    end else begin
```

```
                frame_lost <= frame_lost;
                rx_counter_inc <= 0;
                rx_counter_must_reset_1 <= 0;
                tx_counter_must_reset_2 <= 0;
                edt_strobe_int_pre <= 0;
            end
            if (edt_strobe_int_pre) begin
                edt_strobe_int <= 1;
            end else begin
                edt_strobe_int <= 0;
            end
            if (tx_counter_must_reset_1 | tx_counter_must_reset_2) begin
                tx_counter_reset <= 1;
            end else begin
                tx_counter_reset <= 0;
            end
            if (rx_counter_must_reset_1 | rx_counter_must_reset_2) begin
                rx_counter_reset <= 1;
            end else begin
                rx_counter_reset <= 0;
            end
            rx_register [159:32]    <=  rx_register [127:0];
            rx_register [31:0]  <= rx_data;
        old_write <= write;
    end

    // BUFG - Library Element: Buffer
    BUFG buff_txout (
        .I(tx_out_clk_int_nb),
        .O(tx_out_clk_int)
        );

    BUFG buff_clock32 (
        .I(tx_out_clk_d4),
        .O(tx_out_clk_d4_b));

    // Frequency conversion. The fastest module in EDT-SerDes.
    GTX_8to32 freq_converter (
    .gtxoutclk8(tx_out_clk_int),
    .gtxrx8(rx_data8),
    .gtxrx32(rx_data),
    .gtxtx32(tx_data),
    .gtxtx8(tx_data8),
    .gtxoutclk32(tx_out_clk_d4)
    );

//---------------------------- The GTX Wrapper -----------------------------
    v6_gtxwizard_v1_9 #
    (
        .WRAPPER_SIM_GTXRESET_SPEEDUP   (0)      // Set this to 1 for simulation
    )
    MyTransceiver_i
    (
    //------------------ Receive Ports - RX Data Path interface -----------------
        .GTX0_RXDATA_OUT                (rx_data8),
        .GTX0_RXUSRCLK2_IN              (tx_out_clk_int),
        .GTX0_RXRESET_IN                (rx_rst),
    //----- Receive Ports - RX Driver,OOB signalling,Coupling and Eq.,CDR ------
        //.GTX0_RXEQMIX_IN              (eq_mix[2:0]),        // default 3'b000
        .GTX0_RXN_IN                    (rx_n),
        .GTX0_RXP_IN                    (rx_p),
```

```
    //---------------------- Receive Ports - RX PLL Ports ----------------------
        .GTX0_GTXRXRESET_IN             (gtx_rx_rst),
        .GTX0_MGTREFCLKRX_IN            (mgt_refclk_rx),
        .GTX0_PLLRXRESET_IN             (pll_rx_rst),
        .GTX0_RXPLLLKDET_OUT            (pll_rx_kdet),
        .GTX0_RXRESETDONE_OUT           (rx_reset_done),
    //---------------- Transmit Ports - TX Data Path interface -----------------
        .GTX0_TXDATA_IN                 (tx_data8),
        .GTX0_TXOUTCLK_OUT              (tx_out_clk_int_nb),
        .GTX0_TXUSRCLK2_IN              (tx_out_clk_int),
        .GTX0_TXRESET_IN                (tx_rst),
    //-------------- Transmit Ports - TX Driver and OOB signaling -------------
        //.GTX0_TXDIFFCTRL_IN           (diff_ctrl[3:0]),
        .GTX0_TXN_OUT                   (tx_n),
        .GTX0_TXP_OUT                   (tx_p),
    //---------------------- Transmit Ports - TX PLL Ports ----------------------
        .GTX0_GTXTXRESET_IN             (gtx_tx_rst),
        .GTX0_TXRESETDONE_OUT           (tx_reset_done)
    );
    //----------------------Dedicated GTX Reference Clock Inputs ---------------
    IBUFDS_GTXE1 q4_clk0_refclk_ibufds_i
    (
        .O                      (mgt_refclk_rx[0]),
        .ODIV2                  (mgt_refclk_rx[1]),
        .CEB                    (1'b0),
        .I                      (F6),       // Connect to package pin H6
        .IB                     (F5)        // Connect to package pin H5
    );
endmodule

// ***************************************************************
//  CRC Computation (combinational)
// ===============================================================
// INPUT
// |- 12b data -|- 8b xxx -|- 100 channels -|
//    [119:108]   [107:100]    [99:0]
// OUTPUT:
// |- 12b data -|- 8b CRC -|- 100 channels -|
//    [119:108]   [107:100]    [99:0]
// CompareOK = xxx == CRC
// ***************************************************************
module CRC8(
        input wire  [119:0] DataIn,
        output wire [119:0] DataOut,
        output wire CompareOK,
        input wire  error
    );
  wire [111:0] DATA;
  wire [7:0] CRC;
  assign DATA[111:100] = DataIn[119:108];
  assign DATA[99:0]    = DataIn[99:0];
  assign CRC[0] = DATA[0] ^ DATA[3] ^ DATA[4] ^ DATA[6] ^ DATA[9] ^ DATA[10] ^
     DATA[11] ^ DATA[14] ^ DATA[15] ^ DATA[18] ^ DATA[21] ^ DATA[23] ^ DATA[24] ^
     DATA[25] ^ DATA[31] ^ DATA[32] ^ DATA[33] ^ DATA[34] ^ DATA[38] ^ DATA[39] ^
     DATA[40] ^ DATA[42] ^ DATA[44] ^ DATA[45] ^ DATA[48] ^ DATA[49] ^ DATA[50] ^
     DATA[51] ^ DATA[52] ^ DATA[53] ^ DATA[56] ^ DATA[58] ^ DATA[62] ^ DATA[64] ^
     DATA[65] ^ DATA[67] ^ DATA[69] ^ DATA[71] ^ DATA[74] ^ DATA[78] ^ DATA[79] ^
     DATA[81] ^ DATA[82] ^ DATA[83] ^ DATA[84] ^ DATA[86] ^ DATA[87] ^ DATA[88] ^
     DATA[90] ^ DATA[91] ^ DATA[93] ^ DATA[94] ^ DATA[99] ^ DATA[101] ^ DATA[103] ^
     DATA[104] ^ DATA[105] ^ DATA[106] ^ DATA[107] ^ DATA[109];
  assign CRC[1] = DATA[1] ^ DATA[4] ^ DATA[5] ^ DATA[7] ^ DATA[10] ^ DATA[11] ^
```

```
    DATA[12] ^ DATA[15] ^ DATA[16] ^ DATA[19] ^ DATA[22] ^ DATA[24] ^ DATA[25] ^
    DATA[26] ^ DATA[32] ^ DATA[33] ^ DATA[34] ^ DATA[35] ^ DATA[39] ^ DATA[40] ^
    DATA[41] ^ DATA[43] ^ DATA[45] ^ DATA[46] ^ DATA[49] ^ DATA[50] ^ DATA[51] ^
    DATA[52] ^ DATA[53] ^ DATA[54] ^ DATA[57] ^ DATA[59] ^ DATA[63] ^ DATA[65] ^
    DATA[66] ^ DATA[68] ^ DATA[70] ^ DATA[72] ^ DATA[75] ^ DATA[79] ^ DATA[80] ^
    DATA[82] ^ DATA[83] ^ DATA[84] ^ DATA[85] ^ DATA[87] ^ DATA[88] ^ DATA[89] ^
    DATA[91] ^ DATA[92] ^ DATA[94] ^ DATA[95] ^ DATA[100] ^ DATA[102] ^
    DATA[104] ^ DATA[105] ^ DATA[106] ^ DATA[107] ^ DATA[108] ^ DATA[110];
assign CRC[2] = DATA[2] ^ DATA[5] ^ DATA[6] ^ DATA[8] ^ DATA[11] ^ DATA[12] ^
    DATA[13] ^ DATA[16] ^ DATA[17] ^ DATA[20] ^ DATA[23] ^ DATA[25] ^ DATA[26] ^
    DATA[27] ^ DATA[33] ^ DATA[34] ^ DATA[35] ^ DATA[36] ^ DATA[40] ^ DATA[41] ^
    DATA[42] ^ DATA[44] ^ DATA[46] ^ DATA[47] ^ DATA[50] ^ DATA[51] ^ DATA[52] ^
    DATA[53] ^ DATA[54] ^ DATA[55] ^ DATA[58] ^ DATA[60] ^ DATA[64] ^ DATA[66] ^
    DATA[67] ^ DATA[69] ^ DATA[71] ^ DATA[73] ^ DATA[76] ^ DATA[80] ^ DATA[81] ^
    DATA[83] ^ DATA[84] ^ DATA[85] ^ DATA[86] ^ DATA[88] ^ DATA[89] ^ DATA[90] ^
    DATA[92] ^ DATA[93] ^ DATA[95] ^ DATA[96] ^ DATA[101] ^ DATA[103] ^
    DATA[105] ^ DATA[106] ^ DATA[107] ^ DATA[108] ^ DATA[109] ^ DATA[111];
assign CRC[3] = DATA[3] ^ DATA[6] ^ DATA[7] ^ DATA[9] ^ DATA[12] ^ DATA[13] ^
    DATA[14] ^ DATA[17] ^ DATA[18] ^ DATA[21] ^ DATA[24] ^ DATA[26] ^ DATA[27] ^
    DATA[28] ^ DATA[34] ^ DATA[35] ^ DATA[36] ^ DATA[37] ^ DATA[41] ^ DATA[42] ^
    DATA[43] ^ DATA[45] ^ DATA[47] ^ DATA[48] ^ DATA[51] ^ DATA[52] ^ DATA[53] ^
    DATA[54] ^ DATA[55] ^ DATA[56] ^ DATA[59] ^ DATA[61] ^ DATA[65] ^ DATA[67] ^
    DATA[68] ^ DATA[70] ^ DATA[72] ^ DATA[74] ^ DATA[77] ^ DATA[81] ^ DATA[82] ^
    DATA[84] ^ DATA[85] ^ DATA[86] ^ DATA[87] ^ DATA[89] ^ DATA[90] ^ DATA[91] ^
    DATA[93] ^ DATA[94] ^ DATA[96] ^ DATA[97] ^ DATA[102] ^ DATA[104] ^
    DATA[106] ^ DATA[107] ^ DATA[108] ^ DATA[109] ^ DATA[110];
assign CRC[4] = DATA[0] ^ DATA[3] ^ DATA[6] ^ DATA[7] ^ DATA[8] ^ DATA[9] ^
    DATA[11] ^ DATA[13] ^ DATA[19] ^ DATA[21] ^ DATA[22] ^ DATA[23] ^ DATA[24] ^
    DATA[27] ^ DATA[28] ^ DATA[29] ^ DATA[31] ^ DATA[32] ^ DATA[33] ^ DATA[34] ^
    DATA[35] ^ DATA[36] ^ DATA[37] ^ DATA[39] ^ DATA[40] ^ DATA[43] ^ DATA[45] ^
    DATA[46] ^ DATA[50] ^ DATA[51] ^ DATA[54] ^ DATA[55] ^ DATA[57] ^ DATA[58] ^
    DATA[60] ^ DATA[64] ^ DATA[65] ^ DATA[66] ^ DATA[67] ^ DATA[68] ^ DATA[73] ^
    DATA[74] ^ DATA[75] ^ DATA[79] ^ DATA[81] ^ DATA[84] ^ DATA[85] ^ DATA[92] ^
    DATA[93] ^ DATA[95] ^ DATA[97] ^ DATA[98] ^ DATA[99] ^ DATA[101] ^ DATA[104] ^
    DATA[106] ^ DATA[108] ^ DATA[110] ^ DATA[111];
assign CRC[5] = DATA[0] ^ DATA[1] ^ DATA[3] ^ DATA[6] ^ DATA[7] ^ DATA[8] ^
    DATA[11] ^ DATA[12] ^ DATA[15] ^ DATA[18] ^ DATA[20] ^ DATA[21] ^ DATA[22] ^
    DATA[28] ^ DATA[29] ^ DATA[30] ^ DATA[31] ^ DATA[35] ^ DATA[36] ^ DATA[37] ^
    DATA[39] ^ DATA[41] ^ DATA[42] ^ DATA[45] ^ DATA[46] ^ DATA[47] ^ DATA[48] ^
    DATA[49] ^ DATA[50] ^ DATA[53] ^ DATA[55] ^ DATA[59] ^ DATA[61] ^ DATA[62] ^
    DATA[64] ^ DATA[66] ^ DATA[68] ^ DATA[71] ^ DATA[75] ^ DATA[76] ^ DATA[78] ^
    DATA[79] ^ DATA[80] ^ DATA[81] ^ DATA[83] ^ DATA[84] ^ DATA[85] ^ DATA[87] ^
    DATA[88] ^ DATA[90] ^ DATA[91] ^ DATA[96] ^ DATA[98] ^ DATA[100] ^ DATA[101] ^
    DATA[102] ^ DATA[103] ^ DATA[104] ^ DATA[106] ^ DATA[111];
assign CRC[6] = DATA[1] ^ DATA[2] ^ DATA[4] ^ DATA[7] ^ DATA[8] ^ DATA[9] ^
    DATA[12] ^ DATA[13] ^ DATA[16] ^ DATA[19] ^ DATA[21] ^ DATA[22] ^ DATA[23] ^
    DATA[29] ^ DATA[30] ^ DATA[31] ^ DATA[32] ^ DATA[36] ^ DATA[37] ^ DATA[38] ^
    DATA[40] ^ DATA[42] ^ DATA[43] ^ DATA[46] ^ DATA[47] ^ DATA[48] ^ DATA[49] ^
    DATA[50] ^ DATA[51] ^ DATA[54] ^ DATA[56] ^ DATA[60] ^ DATA[62] ^ DATA[63] ^
    DATA[65] ^ DATA[67] ^ DATA[69] ^ DATA[72] ^ DATA[76] ^ DATA[77] ^ DATA[79] ^
    DATA[80] ^ DATA[81] ^ DATA[82] ^ DATA[84] ^ DATA[85] ^ DATA[86] ^ DATA[88] ^
    DATA[89] ^ DATA[91] ^ DATA[92] ^ DATA[97] ^ DATA[99] ^ DATA[101] ^ DATA[102] ^
    DATA[103] ^ DATA[104] ^ DATA[105] ^ DATA[107];
assign CRC[7] = (DATA[2] ^ DATA[3] ^ DATA[5] ^ DATA[8] ^ DATA[9] ^ DATA[10] ^
    DATA[13] ^ DATA[14] ^ DATA[17] ^ DATA[20] ^ DATA[22] ^ DATA[23] ^ DATA[24] ^
    DATA[30] ^ DATA[31] ^ DATA[32] ^ DATA[33] ^ DATA[37] ^ DATA[38] ^ DATA[39] ^
    DATA[41] ^ DATA[43] ^ DATA[44] ^ DATA[47] ^ DATA[48] ^ DATA[49] ^ DATA[50] ^
    DATA[51] ^ DATA[52] ^ DATA[55] ^ DATA[57] ^ DATA[61] ^ DATA[63] ^ DATA[64] ^
    DATA[66] ^ DATA[68] ^ DATA[70] ^ DATA[73] ^ DATA[77] ^ DATA[78] ^ DATA[80] ^
    DATA[81] ^ DATA[82] ^ DATA[83] ^ DATA[85] ^ DATA[86] ^ DATA[87] ^ DATA[89] ^
    DATA[90] ^ DATA[92] ^ DATA[93] ^ DATA[98] ^ DATA[100] ^ DATA[102] ^
```

```
     DATA[103] ^ DATA[104] ^ DATA[105] ^ DATA[106] ^ DATA[108]) ^ error;
  assign DataOut[119:108] = DataIn[119:108];
  assign DataOut[107:100] = CRC[7:0];
  assign DataOut[99:0] = DataIn[99:0];
  assign CompareOK = (CRC[7:0] == DataIn[107:100]);
endmodule

// ****************************************************************
// Frequency Conversion for GTX bus
// ****************************************************************
module GTX_8to32(
        input wire              gtxoutclk8,     // Clock for 8b bus
        input wire  [7:0]       gtxrx8,         // 8b bus  (received)
        output wire [31:0]      gtxrx32,        // 32b bus (to transmit)
        input wire  [31:0]      gtxtx32,        // 32b bus (received)
        output wire [7:0]       gtxtx8,         // 8b bus (to transmit)
        output wire             gtxoutclk32     // Clock for 32b bus
    );
    reg [1:0]       freq_div;
    reg [31:0]      gtxrx32int;
    reg [31:0]      gtxtx32int;
    reg [7:0]       gtxtx8int;
    initial begin
        freq_div    <=  2'b0;
        gtxrx32int  <= 32'b0;
        gtxtx32int  <= 32'b0;
        gtxtx8int   <= 8'b0;
    end
    assign  gtxoutclk32 = freq_div[1];
    assign  gtxrx32 = gtxrx32int;
    assign  gtxtx8 = gtxtx8int;
    always @ (posedge gtxoutclk8) begin
        freq_div <= freq_div + 1;
        gtxrx32int [31:8]   <=  gtxrx32int [23:0];
        gtxrx32int [7:0]    <= gtxrx8 [7:0];
        gtxtx8int [7:0] <= gtxtx32int [31:24];
        if (freq_div == 2'b10) begin
            gtxtx32int [31:0] <= gtxtx32;
        end else begin
            gtxtx32int [31:8] <= gtxtx32int [23:0];
            gtxtx32int [7:0]    <=  8'b0;
        end
    end
endmodule

// ****************************************************************
// Pseudo-Random Bit Sequence generator (120b LFSR)
// ****************************************************************
module PRBS_generator(
           input wire           clk,
           input wire           reset,
           input wire           next,
           output wire [119:0]  out
           );
    reg [119:0]   lfsr;
    wire          lfsr_feedback;
    reg           old_next;
    assign        out = lfsr;
    assign        lfsr_feedback = lfsr[0] ^ lfsr[106] ^ lfsr[90] ^ lfsr[75] ^
                          lfsr[60] ^ lfsr[45] ^ lfsr[31] ^ lfsr[16];
    initial begin
```

```
            lfsr <= 120'hA89384758349 58ABC347C09DF7A53A;
            old_next <= 0;
    end
    always @ (posedge clk) begin
            if (reset) begin
                    lfsr <= 120'hA89384758349 58ABC347C09DF7A53A;
            end else if (next & ~old_next) begin
                    // shift lfsr
                    lfsr[118:0] <= lfsr[119:1];
                    lfsr[119]      <= lfsr_feedback;
            end
            old_next <= next;
    end
endmodule

// ****************************************************************
// Computes number of different bits in two vectors
// ****************************************************************
module Compare120(
            input wire [119:0] I1,
            input wire [119:0] I2,
            output wire [119:0] O
    );

    wire [119:0] to_count;
    assign to_count = I1 ^ I2;

    wire [119:0] level1a;
    wire [120:0] level1b;
    wire [119:0] level1o;
    assign level1a = to_count & 120'h55555555555555555555555555555555; //0101
    assign level1b[120] = 1'b0;
    assign level1b[119:0] = to_count & 120'hAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA; //1010
    assign level1o = level1a + level1b[120:1];

    wire [119:0] level2a;
    wire [121:0] level2b;
    wire [119:0] level2o;
    assign level2a = level1o & 120'h33333333333333333333333333333333; //0011
    assign level2b[121:120] = 2'b0;
    assign level2b[119:0] = level1o & 120'hCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC; //1100
    assign level2o = level2a + level2b[121:2];

    wire [119:0] level3a;
    wire [123:0] level3b;
    wire [119:0] level3o;
    assign level3a = level2o & 120'h0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F;
    assign level3b[123:120] = 4'b0;
    assign level3b[119:0] = level2o & 120'hF0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0;
    assign level3o = level3a + level3b[123:4];

    wire [119:0] level4a;
    wire [127:0] level4b;
    wire [119:0] level4o;
    assign level4a = level3o & 120'hFF00FF00FF00FF00FF00FF00FF00FF;
    assign level4b[127:120] = 8'b0;
    assign level4b[119:0] = level3o & 120'h00FF00FF00FF00FF00FF00FF00FF00FF00;
    assign level4o = level4a + level4b[127:8];

    wire [119:0] level5a;
    wire [135:0] level5b;
```

```
    wire [119:0] level5o;
    assign level5a = level4o & 120'h00FFFF0000FFFF0000FFFF0000FFFF;
    assign level5b[135:120] = 16'b0;
    assign level5b[119:0] = level4o & 120'hFF0000FFFF0000FFFF0000FFFF0000;
    assign level5o = level5a + level5b[135:16];

    wire [119:0] level6a;
    wire [151:0] level6b;
    wire [119:0] level6o;
    assign level6a = level5o & 120'h000000FFFFFFFF00000000FFFFFFFF;
    assign level6b[151:120] = 32'b0;
    assign level6b[119:0] = level5o & 120'hFFFFFF00000000FFFFFFFF00000000;
    assign level6o = level6a + level6b[151:32];

    wire [119:0] level7a;
    wire [183:0] level7b;
    wire [119:0] level7o;
    assign level7a = level6o & 120'h00000000000000FFFFFFFFFFFFFFFF;
    assign level7b[183:120] = 64'b0;
    assign level7b[119:0] = level6o & 120'hFFFFFFFFFFFFFF0000000000000000;
    assign level7o = level7a + level7b[183:64];

    assign O = level7o;
endmodule
```