



**AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W
KRAKOWIE**

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

KATEDRA METROLOGII I ELEKTRONIKI

Praca dyplomowa

*Projekt uniwersalnej cyfrowej karty pomiarowej z
rekonfigurowalnymi portami we/wy z obsługą standardu
USB 3.0*

*Design of versatile measurement card with reconfigurable
digital IO compatible with USB 3.0 standard*

Autor: *Jan Wołowiec*

Kierunek studiów: *Mikroelektronika w Technice i Medycynie*

Opiekun pracy: *dr inż. Piotr Otfinowski*

Kraków, 2022

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, videogram lub nadanie.”, a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».”, oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobistie i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Serdecznie dziękuję rodzinie i promotorowi za wsparcie okazane w trakcie pisania tej pracy.

Spis treści

1. Wprowadzenie	7
1.1. Cele pracy	7
1.2. Zawartość pracy.....	8
2. Część teoretyczna	9
2.1. Universal Serial Bus (USB).....	9
2.1.1. Topologia	10
2.1.2. Porównanie wersji.....	11
2.1.3. Typy transferów	12
2.2. Analiza rynku	15
2.2.1. Saleae Logic Pro 16	15
2.2.2. DreamSourceLab DSLogic U3Pro16	16
2.2.3. Saleae Logic 16.....	17
2.2.4. DreamSourceLab DSLogic Plus	18
2.2.5. Kingst LA5016.....	18
2.2.6. LeCroy LogicStudio.....	19
2.2.7. Saleae Logic.....	20
2.2.8. ASIX OMEGA.....	20
2.2.9. Podsumowanie dostępnych urządzeń.....	21
2.3. Alternatywne rozwiązania implementacji	22
2.3.1. Samodzielny układ FPGA.....	22
2.3.2. Samodzielny kontroler USB	23
3. Implementacja rozwiązania	27
3.1. Architektura.....	27
3.2. Aplikacja hosta	32
3.2.1. Interfejs graficzny	32

3.2.2. Obsługa USB API.....	34
3.3. Kontroler USB 3.0.....	38
3.3.1. GPIF.....	38
3.3.2. USB.....	41
3.3.3. DMA	43
3.4. Układ FPGA	48
3.4.1. Interfejs FIFO.....	48
3.4.2. Obsługa kanałów.....	50
3.4.3. Parsowanie pakietów.....	52
3.4.4. Informacje debugowe.....	54
4. Weryfikacja rozwiązania	57
4.1. Procedura testowa.....	57
4.2. Poprawność danych	58
4.3. Przepustowość	59
5. Podsumowanie i wnioski.....	63
Wykaz skrótów	65
Bibliografia	67

1. Wprowadzenie

Wraz z rozwojem technologii i zwiększeniem prędkości transmisji danych coraz częściej zachodzi potrzeba analizy sygnałów o częstotliwościach setek MHz. Do akwizycji szybkich sygnałów z powodzeniem można wykorzystać układy FPGA, lecz potrzebny jest równocześnie szybki i niezawodny interfejs który pozwoli nam przesyłać zebrane dane.

Biorąc pod uwagę niezbyt wysoką przepustowość USB 2.0 HighSpeed dla tego zastosowania, lepszym wyborem jest standard USB 3.0. Wraz z większymi możliwościami wzrasta stopnień skomplikowania implementacji takiego rozwiązania i w praktyce najczęściej wykorzystuje się dedykowane kontrolery USB. W związku z tym praca ta skupi się na integracji systemu składającego się z układu FPGA przeznaczonego do akwizycji, kontrolera USB oraz kontrolującego ich programu.

1.1. Cele pracy

Celem pracy dyplomowej było wykonanie projektu karty pomiarowej z cyfrowymi portami wejścia/wyjścia, komunikującej się z komputerem PC za pomocą standardu USB 3.0.

Aby łatwiej śledzić postępy zostały wyznaczone pomniejsze cele:

- Zaprogramowanie układu FPGA Intel Cyclone 10, odpowiadającego za obsługę portów zewnętrznych.
- Zaprogramowanie modułu FX3 SuperSpeed Explorer, zapewniającego komunikację w standardzie USB 3.0 między FPGA a PC.
- Napisanie przykładowej aplikacji sterującej wykorzystującej USB API.
- Opracowanie protokołu komunikacyjnego odpowiednio pomiędzy FPGA i FX3 oraz FX3 i PC.
- Przygotowanie finalnych wersji aplikacji na poszczególne moduły oraz dodanie interfejsu graficznego do prezentowania danych i kontroli urządzenia.

- Przeprowadzenie testów potwierdzających poprawne działanie opracowanego rozwiązania.

1.2. Zawartość pracy

Rozdział 1 zawiera krótkie omówienie zagadnienia tematyki pracy, postawionych celów oraz zawartości rozdziałów.

Rozdział 2 najpierw opisuje aspekty USB pomocne w zrozumieniu problematyki. Następnie omawiane są urządzenia dostępne na rynku pod względem ich podobieństw i różnic, na koniec natomiast rozważane są alternatywne podejścia jakie można było przyjąć zamiast użytej kombinacji **Cypress FX3** i **FPGA Intel Cyclone 10**.

W rozdziale 3 zamieszczony został opis architektury zastosowanego rozwiązania oraz zostały opisane kluczowe aspekty aplikacji powstałych odpowiednio na: PC, Kontroler USB i FPGA.

W rozdziale 4 opisano sposób weryfikacji poprawności otrzymywanych danych jak i przepustowości karty pomiarowej.

W rozdziale 5 podsumowano wykonane prace, ich przebieg oraz rozważono możliwości rozwoju projektu.

Na koniec załączona została bibliografia wraz z wykazem źródeł wykorzystanych w trakcie pisania pracy oraz wykaz skrótów użytych pojęć. W nawiasach kwadratowych dodano odwołania do pozycji w bibliografii.

2. Część teoretyczna

2.1. Universal Serial Bus (USB)

USB jest jednym z najbardziej popularnych interfejsów na świecie. Aktualnie większość urządzeń których używamy posiada interfejs w jakiejś z jego odmian, porty USB znajdziemy m.in. w komputerach, telefonach, drukarkach, kamerach i wielu więcej. Interfejs USB aby odnieść sukces musiał zadowolić zarówno użytkowników jak i deweloperów. Cechy które pozwoliły mu odnieść ten sukces to między innymi:

- niski koszt rozwiązania
- niezawodność
- duża przepustowości
- ewolucja interfejsu wraz z wymaganiami

Dodatkowo z zalet dla użytkowników możemy wymienić:

- łatwość użycia
- brak potrzeby instalowania sterowników
- zasilanie urządzeń i transmisja danych jednym kablem
- możliwość wpinania/wypinania USB w trakcie działania urządzenia,
- zastosowanie jednego kabla do wielu urządzeń

Z kolei w przypadku deweloperów:

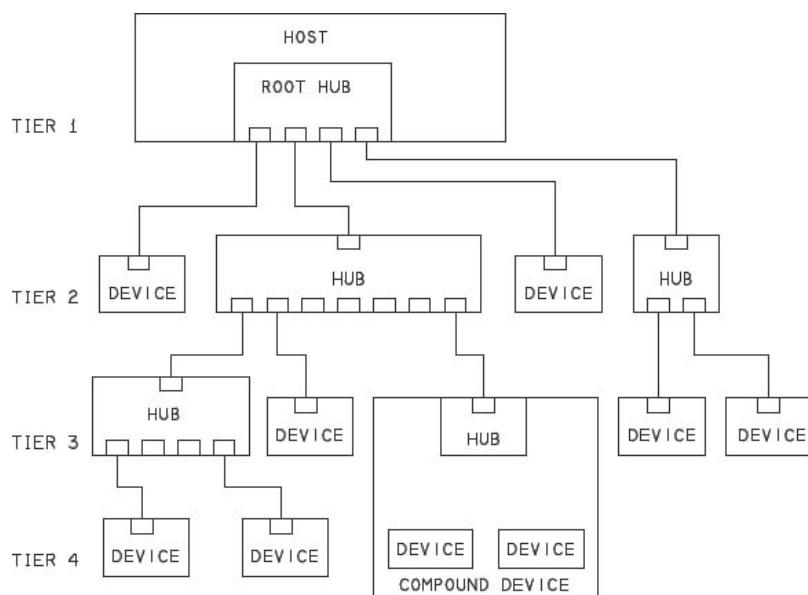
- szeroki wachlarz zastosowań USB z racji różnych typów transferów i wspieranych przedkości
- szerokie wsparcie interfejsu w różnych systemach operacyjnych

2.1.1. Topologia

Protokół USB korzysta z topologii o nazwie “**tiered star topology**” co można by przetłumaczyć jako warstwową topografię gwiazdy. Do komunikacji USB wymagany jest host ze wsparciem USB, hub oraz podłączane urządzenie, oczywiście potrzebujemy również kablemi którymi wszystko połączymy.

kontroler – zarządza urządzeniami w sieci oraz zajmuje się wysyłaniem i odbieraniem danych

hub – pozwala podłączyć wiele urządzeń do komputera, dodatkowo razem z kontrolerem zarządza podłączaniem i odłączaniem urządzeń oraz przesyła dane między urządzeniami i do hosta.



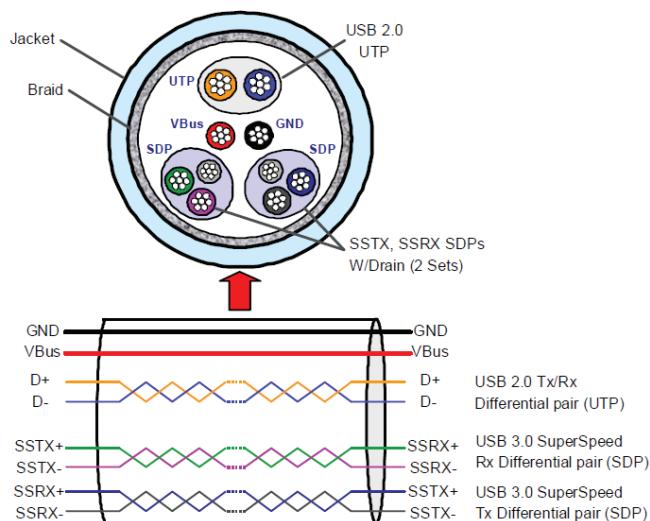
Rys. 2.1. Warstwowa topologia gwiazdy[1]

W tej topologii każde urządzenie komunikuje się z hostem bezpośrednio lub za pośrednictwem hubów. Jak przedstawiono na rysunku 2.1 centrum każdej gwiazdy jest hub od którego rozchodzą się połączenia w kształcie gwiazdy. Dotyczy to również samego hosta który zawiera “root hub”, będący centralnym punktem sieci.

Topologia gwiazdy opisuje jedynie fizyczne połączenia. Biorąc pod uwagę logiczne połączenia, czyli to jak aplikacje i oprogramowanie widzą podłączone urządzenia, huby są niewidzialne z ich perspektywy. Nie ma znaczenia czy urządzenie jest podłączone bezpośrednio, czy poprzez 5 hubów. Należy jednak pamiętać o istniejącym ograniczeniu kontrolera do maksymalnie 127 podłączonych urządzeń oraz 5 hubów szeregowo.

2.1.2. Porównanie wersji

Jednym z powodów dla których USB nadal jest popularny to fakt że protokół ten ciągle ewoluje, starając się sprostać wymaganiom użytkowników i deweloperów. Powstało wiele wersji USB, są one ze sobą wstępnie kompatybilne aby zapewnić jak największą wszechstronność i intuicyjność w obsłudze.



Rys. 2.2. Przekrój kabla USB 3.0[2]

USB 1.0 – Pierwsza wersja USB oferująca prędkości **Low Speed 1.5Mbps**, oraz **Full Speed** wynoszące **12Mbps**. Swoją popularność zyskała w dużej mierze dzięki technologii **“Hot Swap”**, pozwalające podłączać i odłączać urządzenie przy włączonym zasilaniu.

USB 1.1 – W porównaniu do USB 1.0 prędkości transmisji nie zmieniły, natomiast dodano nowy typ transakcji **“interrupt OUT”**.

USB 2.0 – Wraz z rozwojem technologii prędkość **Full Speed 12Mbps** przestała być wystarczająca i zaczęto poszukiwać innego rozwiązania. Okazało się że możliwe jest osiągnięcie prędkości **480Mbps**, zwiększając prędkości **20 razy** względem **Full Speed**, jednocześnie zachowując wstępную kompatybilność. USB do tego czasu nadal wykorzystywało ten sam zestaw 4 pinów w kablu, wykorzystywany począwszy od USB 1.0.

USB 3.0(USB 3.2 Gen 1x1) – Stanowi najbardziej drastyczną zmianę interfejsu USB, daje on 2 pary różnicowe pozwalając dzięki nim osiągać prędkości SuperSpeed sięgające blisko 5Gbps. Dodatkowo dzięki nim transmisja teraz może odbywać się w trybie Full Duplex, zamiast dotychczasowego Half Duplex. Należy jednak zwrócić uwagę, że realna przepustowość wynosi jedynie 4Gbps z powodu wykorzystanego kodowania **8b/10b**, w

przypadku poprzednich wersji wykorzystywane było kodowanie NRZI. Mimo zwiększeniu ilości wyprowadzeń do 9, protokół zachowuje wstępную kompatybilność.

USB 3.1(USB 3.2 Gen 2×1) – Względem USB 3.0 podwaja przepustowość do 10Gbps, jednocześnie zmniejszając over-head dzięki zastosowaniu kodowania 128b/132b.

USB 3.2(USB 3.2 Gen 1×2) – Wykorzystuje drugą parę różnicową portu USB C powodując podwojenie prędkości do poziomu 10Gbps dla kodowania 8b/10b.

USB 3.2(USB 3.2 Gen 2×2) – Wykorzystuje drugą parę różnicową portu USB C powodując kolejne podwojenie prędkości do poziomu 20Gbps jednoczesnym wykorzystaniem kodowania 128b/132b.

Tabela 2.1. Parametry interfejsów USB[3]

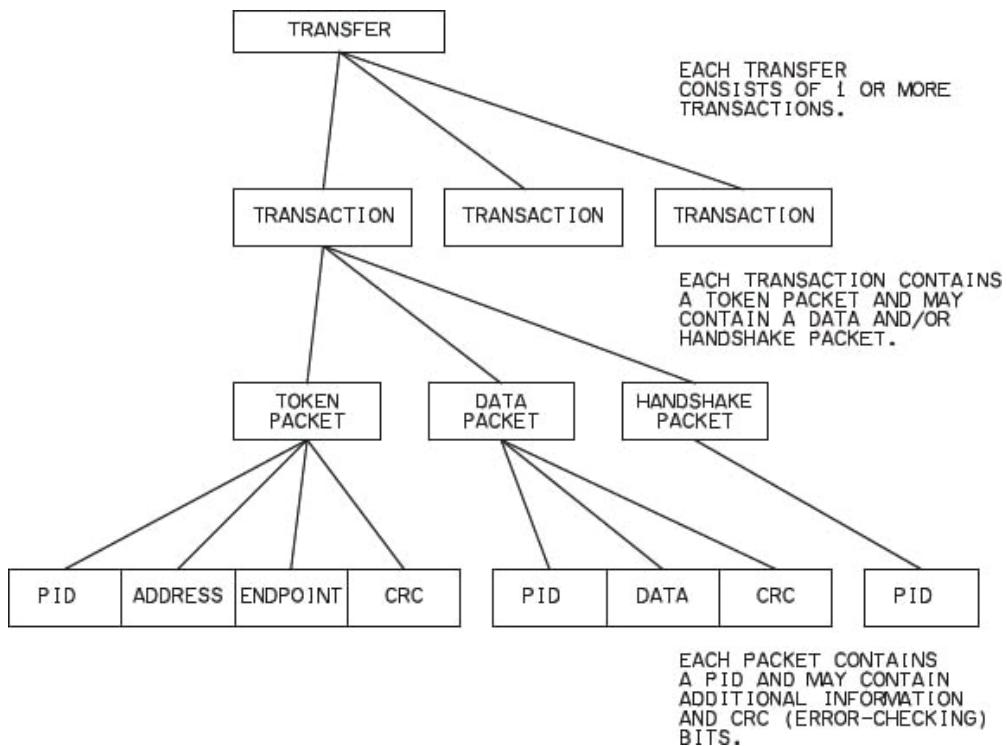
Nazwa	Kodowanie	Pary różnicowe	Prędkość
USB 1.0	NRZI	1 Half-Duplex	1.5Mbps/12Mbps
USB 2.0	NRZI	1 Half-Duplex	480Mbps
USB 3.0(USB 3.2 Gen 1×1)	8b/10b	2 Full-Duplex	5Gbps
USB 3.1(USB 3.2 Gen 2×1)	128b/132b	2 Full-Duplex	10Gbps
USB 3.2(USB 3.2 Gen 1×2)	8b/10b	4 Full-Duplex	10Gbps
USB 3.2(USB 3.2 Gen 2×2)	128b/132b	4 Full-Duplex	20Gbps

2.1.3. Typy transferów

Transfer jest to zbiór jednej lub więcej transakcji, jak zostało to przedstawione na rysunku 2.3. Transakcje składają się z 2 lub 3 pakietów, zależy to od typu transferu jaki wykonujemy ale zazwyczaj polega na przekazaniu danych z hosta do urządzenia, lub na odwrotnie. Typ transakcji najczęściej określa czy mamy do czynienia z transakcją przychodzącej do hosta(IN), czy wychodzącej od hosta(OUT). Szczegóły przedstawiono w tabeli 2.2.

Tabela 2.2. Typy transakcji[1]

Typ transakcji	Źródło danych	Typy transferów które go wykorzystują	Zawartość
IN	urządzenie	wszystkie	dane lub status
OUT	host	wszystkie	dane lub status
Setup	host	Control	zapytanie



Rys. 2.3. Budowa pojedynczego transferu[1]

Transakcja składa się zazwyczaj z 3 pakietów, jedynym wyjątkiem są transakcje izochroniczne, które nie wymagają potwierdzenia do przesyłanych danych.

Token – Host inicjuje za jego pomocą komunikację z wybranym urządzeniem, informując jaki typ transakcji będzie miało miejsce.

Data – Host lub urządzenie może przesyłać dane w tym pakiecie, PID powinien się sekwencyjnie zmieniać po każdej transakcji celem wykrywania zgubionych lub zduplikowanych pakietów.

Handshake – Służy do wysłania informacji o statusie.

Każdy z dostępnych typów transakcji ma spełniać inne zadania, jednakże urządzenie może wspierać jedynie te typy, które najbardziej odpowiadają jego zastosowaniu. Spośród 4 typów transakcji mamy do wyboru:

Control – Pozwalają hostowi odczytać informacje na temat urządzenia oraz odpowiednio je konfigurować.

Bulk – Ten typ jest przeznaczony do przesyłania dużej ilości danych, ale o niskim priorytecie, z tego powodu gdy magistrala jest zajęta będą one wolne, lecz gdy inne urządzenia nie nadają, pozwalają one osiągnąć najwyższe prędkości.

Interrupt – Służą do obsługi urządzeń, które wymagają okresowego odczytu danych, jak i niskiego opóźnienia.

Isochronous – Pozwalają przesyłać dane generowane okresowo, lecz transmisja jest wykonywana bez potwierdzenia oraz bez korekcji błędów, co znaczy że mogą wystąpić okazjonalne błędy transmisji. Głównie używane do streamowania audio i wideo, z racji że mają gwarantowany czas dostarczenia transakcji i opóźnienie, a nie są to krytyczne dane.

Poniżej w tabeli 2.3 przedstawiono szczegóły dla 4 wspomnianych typów transferów.

Tabela 2.3. Zastosowanie typów transferów[1]

Typ transferu	Control	Bulk	Interrupt	Isochronous
Typowe zastosowanie	Identyfikacja i konfiguracja	Drukarka, skaner, dysk	Mysz, klawiatura	Streaming dźwięku, wideo
Wymagane wsparcie urządzenia?	tak	nie	nie	nie
Low speed dozwolone?	tak	nie	tak	nie
Maksymalny rozmiar pakietu; Maksymalny gwarantowany pakiet/okres (Enhanced SuperSpeed).	512; brak	1024; brak	1024; 3 / 125 µs	1024; SuperSpeed: 48 / 125 µs SuperSpeedPlus: 96 / 125 µs
Maksymalny rozmiar pakietu; Maksymalny gwarantowany pakiet/okres(High Speed).	64; brak	512; brak	1024; 3 / 125 µs	1024; 3 / 125 µs
Maksymalny rozmiar pakietu; Maksymalny gwarantowany pakiet/okres(Full Speed).	64; brak	64; brak	64: 1 / ms	1023; 1 / ms
Maksymalny rozmiar pakietu; Maksymalny gwarantowany pakiet/okres(Low Speed).	8; brak	niedozwolone	8; 1 / 10 ms	niedozwolone
Kierunek przepływu danych	IN i OUT	IN lub OUT	IN lub OUT	IN lub OUT
Zarezerwowane pasmo na wszystkie transfery typu	10% Low/Full Speed, 20% High Speed i Enhanced SuperSpeed	brak	90% Low/Full Speed, 80% High Speed i Enhanced SuperSpeed (isochronous i interrupt połączone)	
Message/Stream?	message	stream	stream	stream
Korekcja błędów?	tak	tak	tak	nie
Gwarancja szybkości doręczenia?	nie	nie	nie	tak
Gwarantowane opóźnienie?	nie	nie	tak	tak

2.2. Analiza rynku

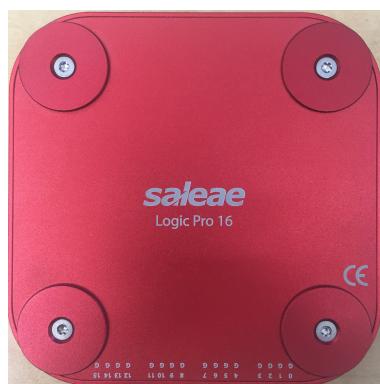
Wykonując przegląd rynku analizatorów logicznych mamy duży wybór urządzeń, mimo podobieństw różnią się one jednak parametrami takimi jak:

- Częstotliwość próbkowania
- Ilość kanałów
- Wbudowana pamięć - do przechowywania próbek
- Tryby działania - buforowanie danych w pamięci przed wysłaniem / streamowanie danych do hosta na bieżąco
- Algorytmy kompresji - w przypadku wolno zmieniających się sygnałów pozwalają zaoszczędzić miejsce w buforze.

Dodatkowo warto również wziąć pod uwagę takie parametry jak maksymalne napięcie wejściowe i dostępne oprogramowanie. W tym porównaniu jednak skupimy się jednak bardziej na wydajności dostępnych rozwiązań.

Pomimo że wszystkie te produkty się między sobą różnią to bardzo często można spotkać podobne konfiguracje sprzętowe pod względem zastosowanego hardware'u.

2.2.1. Saleae Logic Pro 16



Rys. 2.4. Analizator logiczny Saleae Logic Pro 16[4]

Maksymalne parametry w trybie stream	4CH, 500MHz
Maksymalne parametry w trybie buffer	-
Pamięć	-
Konfiguracja sprzętowa	Xilinx Spartan-6 + Cypress FX3

Flagowy produkt popularnej marki Saleae, nie posiada on dodatkowej pamięci, lecz polega na bezpośrednim streamowaniu danych do komputera z pomocą układu Cypress FX3 wspierającego USB 3.0 o przepustowości 5Gbps.

FPGA dodatkowo pozwala na zwiększenie częstotliwości próbkowania w porównaniu do możliwości samego układu FX3 do wartości 500MHz dla 4 kanałów jednocześnie.

Z powodu ograniczeń komunikacji pomiędzy FPGA a FX3, maksymalna przepustowość interfejsu FIFO ogranicza się do 100MHz dla 32-bitowych danych, co przelicza się na przepustowość na poziomie 3.2Gbps, mimo to analizator ten wykorzystuje maksymalnie 2Gbps w trybie 500MHz 4CH.

2.2.2. DreamSourceLab DSLogic U3Pro16



Rys. 2.5. Analizator logiczny DreamSourceLab DSLogic U3Pro16[5]

Maksymalne parametry w trybie stream	3CH, 1GHz
Maksymalne parametry w trybie buffer	8CH, 1GHz
Pamięć	256MiB
Konfiguracja sprzętowa	Xilinx Spartan-6 + Cypress FX3

Produkt DreamSourceLab mimo że mniej popularny niż Saleae, okazuje się oferować lepszą wydajność przy niższej cenie niż konkurent.

Konfiguracja sprzętowa jest bardzo podobna do tej z Saleae Logic Pro 16, jedną różnicą jest dodatkowa pamięć do buforowania próbek. Pomimo tego już w trybie streamowania, można uzyskać 1GHz dla 3 kanałów co w pełni pozwala wykorzystać możliwości FIFO układu Cypress FX3.

Osiagnie częstotliwości próbkowania 1GHz nie jest jednak prostym zadaniem i prawdopodobnie wykorzystano porty SelectIO umożliwiające uzyskanie transferów do 1,080 Mb/s dla zastosowanego FPGA[6].

Zgodnie z dokumentacją bloku High Speed SelectIO[7] wykonuje on serializację przesyłanych danych, w nowszych układach FPGA osiągającą prędkości do 1600Mbps z 8 lub 4-krotnym współczynnikiem serializacji.

Wracając do różnic w konfiguracji sprzętowej, dzięki zastosowaniu dodatkowej pamięci możliwe jest uzyskanie częstotliwości 1GHz dla 8 kanałów, gdy próbki są zapisywane do bufora. Aby bardziej optymalnie przechowywać dane (na przykład dla wolno zmieniających się sygnałów) wykorzystywana jest również kompresja, co pozwala jeszcze bardziej zwiększyć możliwy czas akwizycji danych.

2.2.3. Saleae Logic 16



Rys. 2.6. Analizator logiczny Saleae Logic 16[8]

Maksymalne parametry w trybie stream	3CH, 100MHz
Maksymalne parametry w trybie buffer	-
Pamięć	-
Konfiguracja sprzętowa	Xilinx Spartan-3A + Cypress FX2

W porównaniu do dużego brata Saleae Logic 16 działa w podobny sposób streamując dane przez interfejs USB bez wykorzystania bufora. Główną różnicą jest tu zastosowanie kontrolera USB Cypress FX2, wykorzystującego jedynie interfejs USB 2.0 z przepustowością do 480Mbps.

Zastosowanie FPGA nadal pozwala na większą kontrolę nad próbkowaniem danych, lecz z racji ograniczeń USB 2.0, producent ograniczył nas do 100MHz dla 3 kanałów, co daje 300Mbps prawdopodobnie wynikające z faktycznych osiągów USB 2.0 w praktyce.

2.2.4. DreamSourceLab DSLogic Plus



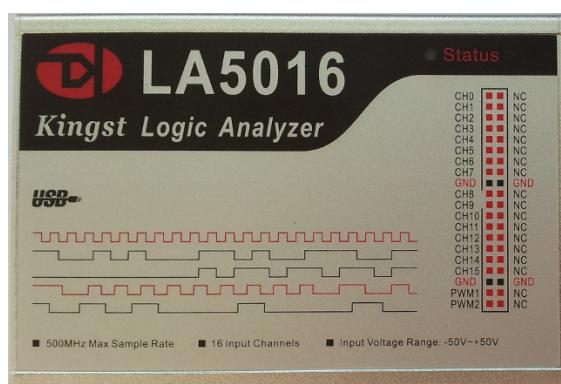
Rys. 2.7. Analizator logiczny DreamSourceLab DSLogic Plus[9]

Maksymalne parametry w trybie stream	3CH, 100MHz
Maksymalne parametry w trybie buffer	4CH, 400MHz
Pamięć	256MiB
Konfiguracja sprzętowa	Xilinx Spartan-6 + Cypress FX2

Podobnie jak porównując Saleae Logic Pro 16 i DreamSourceLab DSLogic U3Pro16, tutaj widać analogię między tym układem a Saleae Logic 16, gdzie główną różnicą jest zastosowanie dodatkowej pamięci.

Mimo takiego samego ograniczenia przez USB 2.0 do 3 kanałów 100MHz, w trybie buforowania analizator osiąga częstotliwości próbkowania rzędu 400MHz dla 4 kanałów, dodatkowo kompresując zebrane dane w buforze.

2.2.5. Kingst LA5016



Rys. 2.8. Analizator logiczny Kingst LA5016[10]

Maksymalne parametry w trybie stream	300Mbps (adekwatne do 3CH, 100MHz)
Maksymalne parametry w trybie buffer	16CH, 500MHz
Pamięć	256MiB
Konfiguracja sprzętowa	Intel Cyclone IV + Cypress FX2

Kingst korzysta z nieco innego hardware'u używając FPGA Intel Cyclone IV, lecz Cypress FX2 ponownie stanowi wąskie gardło. Wykorzystywane jest buforowanie danych pozwalające zbierać dane z częstotliwościami próbkowania rzędu 500MHz.

2.2.6. LeCroy LogicStudio



Rys. 2.9. Analizator logiczny LeCroy LogicStudio[11]

Maksymalne parametry w trybie stream	Brak danych
Maksymalne parametry w trybie buffer	8CH, 1GHz
Pamięć	40KB
Konfiguracja sprzętowa	Xilinx Spartan-6 + Cypress FX2

Analizator marki LeCroy jest jednym z bardziej specyficznych produktów, gdyż mimo wolnego interfejsu USB 2.0 możliwe jest próbkowanie danych z częstotliwością 1GHz. Niestety wykorzystanie bardzo małego bufora w postaci 40KB poddaje wątpliwość praktyczne zastosowanie takiego analizatora.

2.2.7. Saleae Logic



Rys. 2.10. Analizator logiczny Saleae Logic[12]

Maksymalne parametry w trybie stream	8CH, 24MHz
Maksymalne parametry w trybie buffer	-
Pamięć	-
Konfiguracja sprzętowa	Cypress FX2

Jeden z podstawowych modeli Saleae, na ten moment już dyskontynuowany. Nie wyłamyując się z szeregu analizatorów Saleae również nie posiada wbudowanej pamięci. Dodatkowo w porównaniu do wcześniej analizowanych modeli nie wykorzystuje też FPGA, lecz bezpośrednio używa układu Cypress FX2 do próbkowania danych, co niestety ogranicza nas jedynie do częstotliwości próbkowania 24MHz. Z racji na prostotę budowy, powstało wiele klonów tego urządzenia i można łatwo kupić jego chińskie podróbki.

2.2.8. ASIX OMEGA



Rys. 2.11. Analizator logiczny ASIX OMEGA[13]

Maksymalne parametry w trybie stream	Ograniczone przez przepustowość układu FTDI
Maksymalne parametry w trybie buffer	8CH, 400MHz
Pamięć	64MiB
Konfiguracja sprzętowa	Xilinx Spartan-3A + FTDI FT232HL

Jest to jeden z nielicznych analizatorów wykorzystujących układ FTDI do komunikacji z komputerem zamiast Cypress FX2. Nie wiadomo czy była to dobra decyzja, gdyż według specyfikacji technicznej układ FTDI może osiągać maksymalną przepustowość interfejsu FIFO w granicy 40MB/s co przelicza się na 320Mbps wypadając gorzej niż układ FX2.

Aby nadrobić niską przepustowość kontrolera USB 2.0, wykorzystany został bufor wraz z algorytmami kompresji.

2.2.9. Podsumowanie dostępnych urządzeń

Porównując między sobą przeanalizowane urządzenia widać pewne powtarzające się zależności:

- Zdecydowana większość analizatorów wykorzystuje układy FPGA w celach akwizycji danych. Jest to szczególnie ważne przy częstotliwościach próbkowania ponad 100MHz. Jedyne urządzenia na rynku działające wyłącznie z kontrolerem USB to praktycznie tylko układy Cypress FX2, które ze względu na niską cenę mogły usprawiedliwić częstotliwości próbkowania rzędu 24MHz. W przypadku kontrolerów Cypress FX3 zawsze były one parowane z układem FPGA, by wykorzystać w pełni ich potencjał gdyż maksymalna częstotliwość próbkowania w przeciwnym wypadku ograniczyłaby się do 100MHz, marnując możliwy potencjał i podwyższając cenę.
- Ze względu na przepustowość USB 3.0 analizatory wykorzystujące ten standard są w stanie działać streamując dane bezpośrednio, lecz oczywiście w przypadku gdyby była potrzebna akwizycji wielu kanałów jednoczenie użycie bufora może być konieczne.
- W przypadku urządzeń wykorzystujących USB 2.0 użycie buforów i algorytmów kompresji jest dużo bardziej krytyczne, z racji że bezpośrednie streamowanie danych zazwyczaj pozwala w takim wypadku maksymalnie na użycie 3 kanałów z próbkowaniem 100MHz.

2.3. Alternatywne rozwiązania implementacji

Jak już mogliśmy zobaczyć w podrozdziale 2.2, najpopularniejszym i zapewniającym największą wydajność rozwiązaniem jest połączenie kontrolera USB 3.0 z układem FPGA. Taka właśnie kombinacja została wykorzystana w tej pracy, lecz zanim zaczniemy ją omawiać warto zastanowić się jakie są inne możliwości wykonania analizatora stanów logicznych.

2.3.1. Samodzielny układ FPGA

2.3.1.1. Szybkie interfejsy komunikacji

Największym problemem przy wykorzystaniu wyłącznie układu FPGA, jest implementacja szybkiej komunikacji z komputerem. Własnoręczna implementacja protokołu USB 3.0 pochłonęłaby niewspółmiernie więcej wysiłku w porównaniu do samej akwizycji danych oraz wymagała użycia bardziej zaawansowanej rodziny układów FPGA z odpowiednimi portami, podnosząc cenę takiego rozwiązania. Aby zagwarantować kompatybilność ze standardem USB 3.0 najlepszym rozwiązaniem byłoby wykorzystanie dedykowanego IP Core, co zazwyczaj nie jest tanim rozwiązaniem.

Kolejną opcją umożliwiającą szybkie prędkości transmisji jest wykorzystanie interfejsu PCIe, w tym wypadku sytuacja wygląda lepiej gdyż producenci jak np. Xilinx udostępniają za darmo IP Core zapewniające obsługę tego interfejsu. Poniżej zamieszczony został wycinek z dokumentacji producenta dający pogląd na możliwości tego protokołu. Dla najszybszej opcji PCIe 2.0 x8 i uwzględnieniu kodowania 8b/10b, otrzymujemy przepustowość na poziomie 32Gbps.

Product Name	User Interface Width	Supported Lane Widths
1-lane at 2.5 Gb/s, 5.0 Gb/s	64	x1
2-lane at 2.5 Gb/s, 5.0 Gb/s	64	x1, x2 ⁽¹⁾
4-lane at 2.5 Gb/s, 5.0 Gb/s	64, 128	x1, x2, x4 ^{(1),(2)}
8-lane at 2.5 Gb/s, 5.0 Gb/s	64, 128	x1, x2, x4, x8 ^{(1),(3)}

Rys. 2.12. Podsumowanie możliwości IP Core dostarczanego przez Xilinx[14]

Mimo lepszych osiągów niż USB 3.0 należy jednak wziąć pod uwagę znaczne ograniczenie mobilności takiego rozwiązania i konieczności zaopatrzenia w komputer z wolnym portem PCIe. Podobnie jak w przypadku USB wymagane jest również zastosowanie bardziej zaawansowanej rodziny FPGA z odpowiednim złączem PCIe.

2.3.1.2. Buforowanie danych

Aby ominąć problem implementacji szybkiego interfejsu komunikacji w FPGA, można wykorzystać wolniejszy interfejs komunikacji, zamiast tego dodając bufor pamięci do przechowywanie próbek. Podobne podejście było zastosowane w przypadku analizatorów wykorzystujących układ Cypress FX2, aby nadrobić niską przepustowość USB 2.0. Oczywiście w tym przypadku kontrola USB odbywały się w FPGA, w tym celu można na przykład wykorzystać IP Core "USB 2.0 Function Core" dostępny na platformie OpenCores implementujący ten interfejs. Z racji prędkości interfejsu wymagane jest również użycie odpowiedniego transceivera.

Jedną z opcji które można rozważyć jest wykorzystanie jeszcze prostszego interfejsu, jak na przykład port szeregowy(UART). Skutkowałoby to dłuższymi czasami przesyłu danych, co jednak nie wpływa na działanie samego analizatora.

Właśnie takie podejście zastosowali twórcy oprogramowania LiteScope[15] służącego do wykorzystania prostych rodzin układów FPGA jako analizatorów logicznych.

```
IO peek and poke with LiteScopeIO.  
Logic analyser with LiteScopeAnalyzer:  
    Subsampling.  
    Data storage in Block RAM.  
    Configurable triggers.  
Bridges:  
    UART <--> Wishbone (provided by LiteX)  
    Ethernet <--> Wishbone ("Etherbone") (provided by LiteEth)  
    PCIe <--> Wishbone (provided by LitePCIe)  
Exports formats: .vcd, .sr(sigrok), .csv, .py, etc...
```

Listing 2.1. Funkcjonalności oprogramowania LiteScope[15]

Jak widać w listingu 2.1, twórcy zdecydowali się na wsparcie częściej spotykanych podzespołów w układach FPGA jak FPGA i Ethernet, jak również PCIe. Wykorzystanie takiego rozwiązania jeśli mamy do dyspozycji sam podstawowy układ FPGA, jest bardzo opłacalne biorąc pod uwagę ceny niektórych analizatorów stanów logicznych.

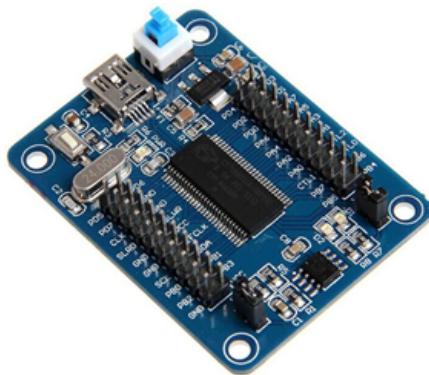
2.3.2. Samodzielny kontroler USB

Podejściem odwrotnym do rozważanego w rozdziale 2.3.1 byłaby rezygnacja z FPGA na rzecz samego kontrolera USB. W takim przypadku nie musimy się martwić o obsługę USB, jako że zajmie się tym kontroler lecz tracimy możliwość wykorzystania zewnętrznej pamięci jako buforów i kompresji danych oraz możliwości próbkowania są ograniczone do wartości jakie wspiera interfejs tych układów. Dane będą streamowane bezpośrednio ze wszystkich pinów

tego interfejsu, w przeciwieństwie do FPGA które pozwalało potencjalnie zmniejszyć ilość próbkowanych kanałów, zwiększając częstotliwość próbkowania dla pozostałych.

2.3.2.1. Cypress FX2

FX2 jest jednym z najczęściej wykorzystywanych układów zarówno samodzielnie, jak i w połączeniu z FPGA. Zawdzięcza to przede wszystkim swojej niskiej cenie i dużych możliwościach konfiguracji. Jeśli chcemy maksymalnie ograniczyć koszty i korzystać wyłącznie z samego układu na stronie producenta dostępny jest artykuł pod tytułem **“Cypress EZ-USB FX2LP-based Logic Analyzer using Open Source sigrok PulseView”**[16] w którym pokazano jak w prosty sposób skonfigurować układ, aby współpracował z oprogramowaniem **sigrok**. Jest to otwarto źródłowe oprogramowanie do analizy sygnałów, wspierające wiele popularnych analizatorów.

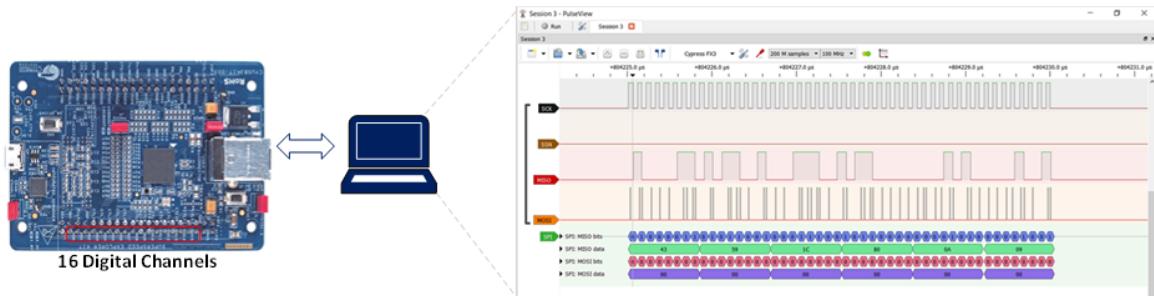


Rys. 2.13. Płytkę ewaluacyjną Lcsoft CY7C68013A Mini Board[16]

Płytkę po takiej konfiguracji pozwala na obsługę **8 kanałów** z częstotliwością próbkowania **24MHz**, takie same parametry oferują często podstawowe modele analizatorów jak np wspomniany Saleae Logic. Z racji że układ ten nie posiada żadnych buforów ani zabezpieczeń ESD, należy uważać przy podłączaniu go do innych urządzeń aby nie został uszkodzony. Zalecane jest, aby co najmniej dodać rezystory pomiędzy testowanym układem, a naszym analizatorem.

2.3.2.2. Cypress FX3

FX3 jako zarówno droższy i bardziej zaawansowany następcą FX2 jest wykorzystywany jedynie w analizatorach z wyższej półki, w których zawsze też dodawany jest układ FPGA, aby wykorzystać pełnię możliwości tego kontrolera USB 3.0. Nic nie stoi jednak na przeszkodzie, aby wykorzystać go bezpośrednio jako analizator. Tak jak i w przypadku FX2 dostępny jest artykuł pod tytułem **“EZ-USB™ FX3 Explorer kit as 16-channel 100 MHz logic analyzer with sigrok PulseView”**[17] opisujący konfigurację układu celem komunikacji z oprogramowaniem **sigrok**.

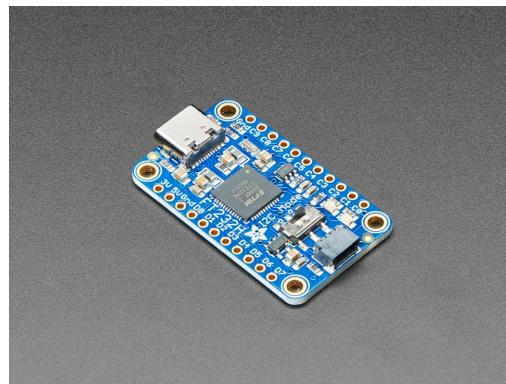


Rys. 2.14. Płytkę ewaluacyjną CYUSB3KIT-003[17]

Analizator wspiera **16 kanałów**, natomiast częstotliwość próbkowania wynosi **100MHz**. Parametry te z pewnością nie mogą rywalizować z analizatorami z górnej półki, lecz biorąc pod uwagę cenę płytka ewaluacyjnej, wypada ona dobrze na tle konkurencji, szczególnie biorąc po uwagę że na dzień pisania tej pracy można ją kupić do 300 PLN, w porównaniu do np. Saleae Logic 8 kosztującego ponad 2500PLN i oferującego próbkowanie **100MHz 3 kanałów** w porównaniu do **16 kanałów** dla płytka FX3. Oczywiście jako że jest to płytka ewaluacyjna, należy pamiętać o zabezpieczeniu pinów wejściowych przed wpięciem się do innego układu.

2.3.2.3. FT232H

Układ od FTDI jest wielofunkcyjnym układem pozwalającym komputerowi na komunikację protokołami I2C, UART, SPI oraz obsługę przez interfejs USB 2.0. Istnieją również dodatkowe tryby FIFO do komunikacji z FPGA lub mikrokontrolerem oraz tryb Bit-Bang, w którym możemy wpisywać i odczytywać surowe dane z portów urządzenia.



Rys. 2.15. Płytkę ewaluacyjną FT232H[18]

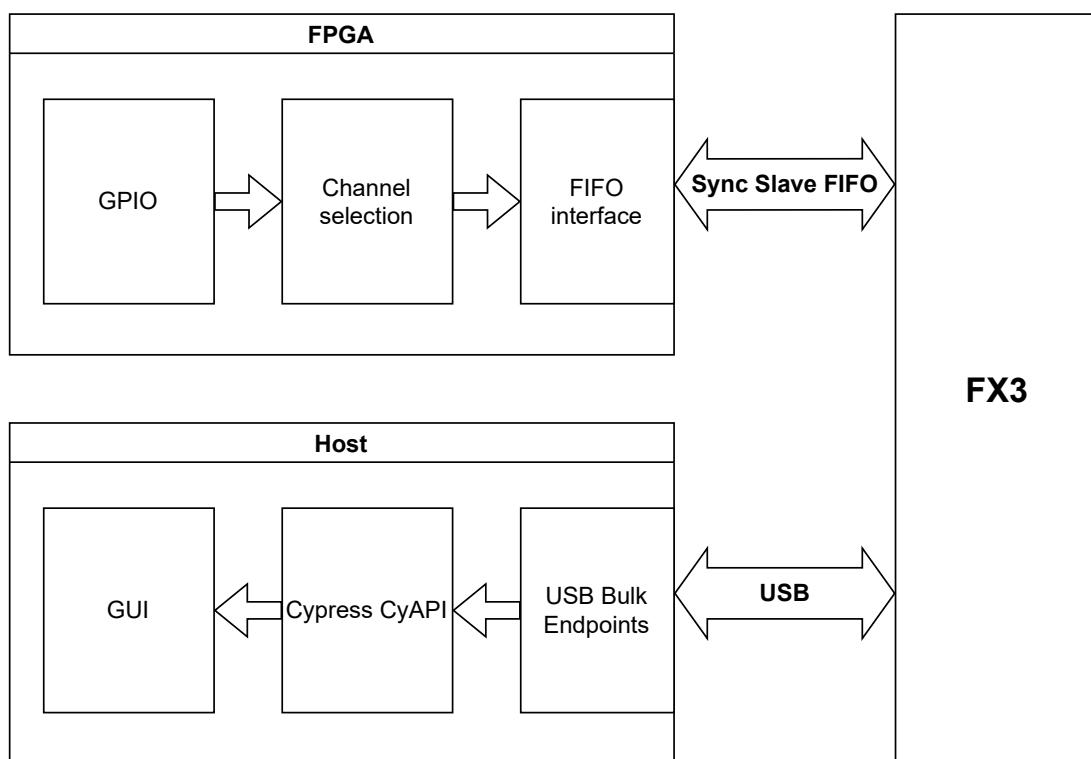
Jeśli chcemy wykorzystać ten układ jako analizator najbardziej przydatny będzie właśnie tryb Bit-Bang, odczytując dane z 8 pinów, daje nam 8 kanałowy analizator o częstotliwości próbkowania w okolicach 10MHz. Ważne jest aby nie korzystać z wersji FT232R, gdyż posiada on problem z jitterem zegara, co czyni uzyskane dane bezużyteczne.

Biorąc pod uwagę że wersja FT232R nie nadaje się do wykorzystania jako analizator, jesteśmy zmuszeni zakupić droższą wersję H. W takim wypadku dużo lepszym wyborem wydaje się Cypress FX2, który nie dość że jest tańszy to zapewnia częstotliwość próbkowania na poziomie 24MHz. Nie musimy też się martwić o to że sterownik zablokuje nasz układ jak w przypadku akcji FTDIgate, gdzie producent wydał sterownik blokujący nieoryginalne układy.

3. Implementacja rozwiązania

3.1. Architektura

Finalny efektem prac jest urządzenie składające się z 3 części: **FPGA**, **Kontrolera FX3** i **Aplikacji Hosta**. Na rysunku 3.1 przedstawiono poglądowo przepływ danych z FPGA do Hosta.



Rys. 3.1. Ogólna architektura rozwiązania

Płytkę CYUSB3KIT-003 z układem Cypress FX3 posiada wyprowadzone 2 złącza 2x20 pinów znajdujące się po obu stronach płytka. Płytkę ewaluacyjną z FPGA Intel® Cyclone® 10 LP, posiada z kolei 1 złącze 2x20 pinów oraz złącze do nakładek Arduino. Celem połączenia obu układów został wykorzystany niestandardowy adapter wykonany na zamówienie. Host będący komputerem PC został podłączony do płytki CYUSB3KIT-003 za pomocą przewodu USB 3.0.

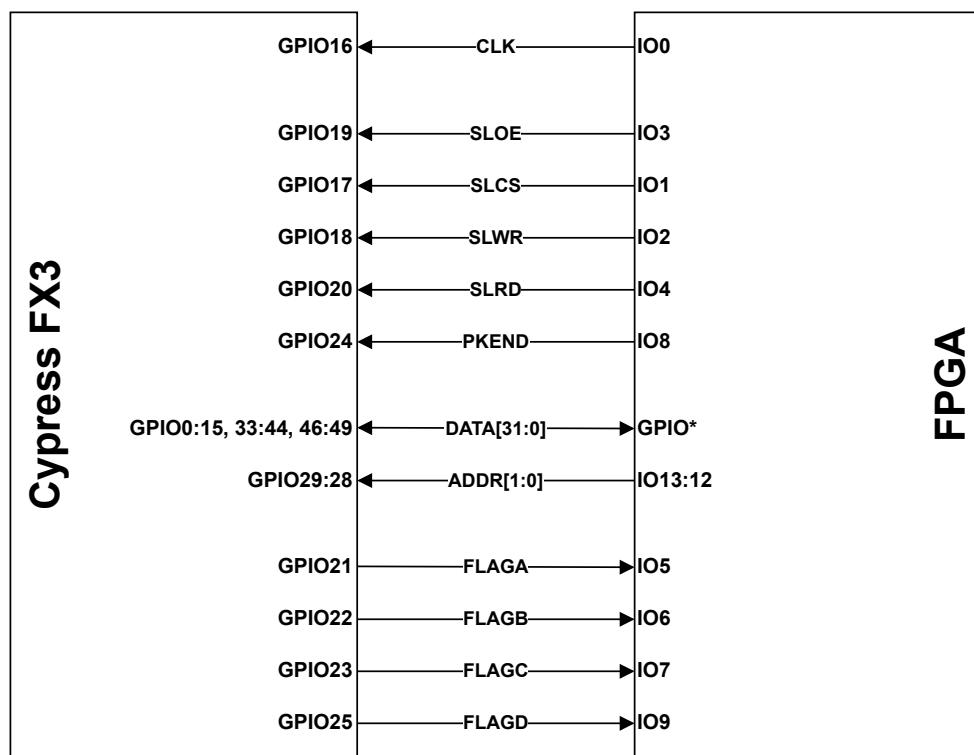
Tabela 3.1. Lewe złącze płytka CYUSB3KIT-003

Pin No.	Pin Name	GPIO	GPIF	FPGA	Pin No.	Pin Pin Name	GPIO	GPIF	FPGA
1	V3P3	-	-	-	2	V1P2	-	-	-
3	INT_N_CTL15	INT#	-	GPIO1/L16	4	VIO	-	-	-
5	NC	-	-	-	6	DQ31	GPIO49	Databus	GPIO4/P16
7	PMODE_2	GPIO32	-	-	8	DQ30	GPIO48	Databus	GPIO6/N16
9	PMODE_1	GPIO31	-	-	10	DQ29	GPIO47	Databus	GPIO8/N14
11	PMODE_0	GPIO30	-	-	12	DQ28	GPIO46	Databus	GPIO10/N8
13	GND	-	-	-	14	GPIO45	GPIO45	-	-
15	GND	-	-	-	16	DQ27	GPIO44	Databus	GPIO12/M8
17	GND	-	-	-	18	DQ26	GPIO43	Databus	GPIO14/R7
19	MOSI_RX	GPIO56	-	-	20	DQ25	GPIO42	Databus	GPIO16/L7
21	MISO_TX	GPIO55	-	-	22	DQ24	GPIO41	Databus	GPIO18/R6
23	SSN_CTS	GPIO54	-	-	24	DQ23	GPIO40	Databus	GPIO20/T2
25	GND	-	-	-	26	DQ22	GPIO39	Databus	GPIO22/R5
27	SCK_RTS	GPIO53	-	-	28	DQ21	GPIO38	Databus	GPIO24/N5
29	GND	-	-	-	30	DQ20	GPIO37	Databus	GPIO26/R4
31	I2S_WS	GPIO52	-	-	32	DQ19	GPIO36	Databus	GPIO28/N3
33	I2S_SD	GPIO51	-	-	34	DQ18	GPIO35	Databus	GPIO30/R3
35	I2S_MCLK	GPIO57	-	-	36	DQ17	GPIO34	Databus	GPIO32/P6
37	I2S_CLK	GPIO50	-	-	38	DQ16	GPIO33	Databus	GPIO34/P1
39	GND	-	-	-	40	GND	-	-	-

Tabela 3.2. Prawe złącze płytka CYUSB3KIT-003

Pin No.	Pin Name	GPIO	GPIF	FPGA	Pin No.	Pin Name	GPIO	GPIF	FPGA
1	V3P3	-	-	-	2	V1P2	-	-	-
3	USB3_VBUS	-	-	VCC_5V_AR1	4	USB3_VBUS	-	-	VCC_5V_AR1
5	GND	-	-	-	6	GND	-	-	GND
7	CTL12	GPIO29	Addr0	IO13/L1	8	DQ15	GPIO15	Databus	GPIO5/R16
9	CTL11	GPIO28	Addr1	IO12/L2	10	DQ14	GPIO14	Databus	GPIO7/N15
11	CTL10	GPIO27	Addr2	IO11/K1	12	DQ13	GPIO13	Databus	GPIO9/P15
13	CTL9	GPIO26	Addr3	IO10/L4	14	DQ12	GPIO12	Databus	GPIO11/P8
15	CTL8	GPIO25	FLAGD	IO9/K5	16	DQ11	GPIO11	Databus	GPIO13/L8
17	CTL7	GPIO24	PKEND	IO8/K2	18	DQ10	GPIO10	Databus	GPIO15/T7
19	CTL6	GPIO23	FLAGC	IO7/J1	20	DQ9	GPIO9	Databus	GPIO17/M7
21	CTL5	GPIO22	FLAGB	IO6/J2	22	DQ8	GPIO8	Databus	GPIO19/T6
23	CTL4	GPIO21	FLAGA	IO5/G1	24	DQ7	GPIO7	Databus	GPIO21/M6
25	CTL3	GPIO20	SLRD	IO4/G2	26	DQ6	GPIO6	Databus	GPIO23/T5
27	CTL2	GPIO19	SLOE	IO3/D1	28	DQ5	GPIO5	Databus	GPIO25/N6
29	CTL1	GPIO18	SLWR	IO2/F3	30	DQ4	GPIO4	Databus	GPIO27/T4
31	CTL0	GPIO17	SLCS	IO1/C2	32	DQ3	GPIO3	Databus	GPIO29/P3
33	GND	-	-	-	34	DQ2	GPIO2	Databus	GPIO31/T3
35	PCLK	GPIO16	CLK	IO0/B1	36	DQ1	GPIO1	Databus	GPIO33/P2
37	GND	-	-	-	38	DQ0	GPIO0	Databus	GPIO35/R1
39	I2C_SDA_FX3	-	-	-	40	I2C_SCL_FX3	-	-	-

W tabelach 3.1 i 3.2 przedstawiono podsumowanie połączenia elektrycznego między układami. Zaczynając od numerów i nazw pinów nadrukowanych na płytce kontrolera FX3 wypisano zgodnie z dokumentacją wewnętrzne mapowanie GPIO. W kolumnach opisanych jako GPIF zapisana jest funkcjonalność danych GPIO skonfigurowanych poprzez oprogramowanie **GPIF II Designer**. Następnie w kolumnach FPGA znajduje się mapowanie adaptera pinów płytki FX3 na piny płytka FPGA, oraz ich nazewnictwa w samym układzie Intel® Cyclone® 10 LP.



Rys. 3.2. Diagram połączeń interfejsu **Slave Fifo** między Cypress FX3, a FPGA

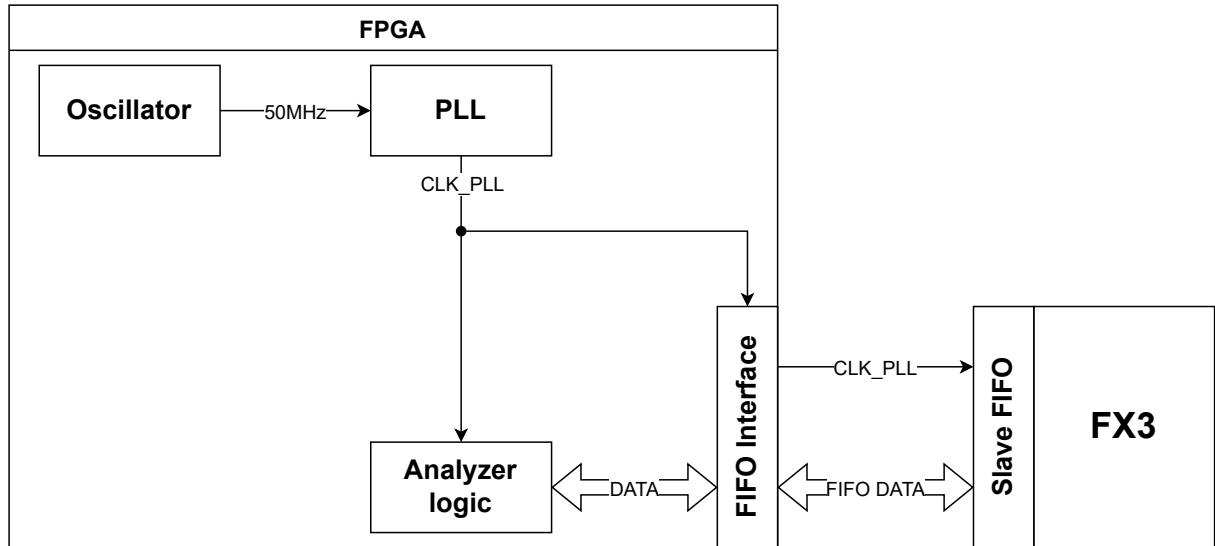
Do komunikacji między kontrolerem USB, a FPGA wykorzystywany jest interfejs **Slave Fifo**, którego połączenia przedstawiono na rysunku 3.2. Linię DATA po stronie FPGA została oznaczona symbolem **GPIO***, gdyż występuje tam wiele pojedynczych pinów, szczegóły przypisania tego połączenia są dostępne w tabelach 3.1 oraz 3.2.

Interfejs ten za pomocą flag symbolizuje gotowość do wysyłania/odbierania danych, po czym FPGA działający jako master może wykonać odpowiednie operacje read i write. W tabeli 3.3 przedstawiono wyjaśnienie wykorzystywanych sygnałów. Wszystkie sygnały są aktywne stanem niskim.

Tabela 3.3. Sygnały interfejsu FIFO

Nazwa sygnału	Opis sygnału
CLK	Clock - Zegar taktujący interfejs FIFO
SLOE	Output Enable - Pozwala FX3 sterować linią DATA, musi być ustawione w stan aktywny podczas transakcji odczytu.
SLCS	Chip Select - Ustawiony w stan aktywny, aby umożliwić korzystanie z interfejsu FIFO.
SLWR	Write Strobe - Ustawiony w stan aktywny, aby umożliwić wykonanie zapisu do FIFO.
SLRD	Read Strobe - Ustawiony w stan aktywny, aby umożliwić wykonanie odczytu z FIFO.
PKTEND	Packet End - Ustawiony w stan aktywny, przy transmisji krótkiego pakietu do hosta.
DATA	Data bus - 32 bitowa linia danych służąca do przesyłu i odbioru danych.
ADDR	Address Bus - 2 bitowa linia adresowania służąca do wyboru aktywnego wątku. Wykorzystywany jest 1 wątek w trakcie odczytu oraz 2 naprzemianie w trakcie zapisu.
FLAGA	Thread 1 DMA Ready - DMA jest gotowe do działania.
FLAGB	Thread 1 DMA WaterMark - flaga ustawiana w stan aktywny, gdy w buforze jest mniej niż 6 wolnych 4-bajtowych słów do zapisu. Flaga wskazuje czy bufor jest pełny jeśli następuje zapis do socket'u.
FLAGC	Thread 3 DMA Ready - DMA jest gotowe do działania.
FLAGD	Thread 3 DMA WaterMark - flaga ustawiana w stan aktywny, gdy w buforze jest mniej niż 6 4-bajtowych słów do odczytu. Flaga wskazuje czy bufor jest pusty jeśli następuje odczyt z socket'u.

Interfejs Fifo jest taktowany zegarem generowanym przez układ FPGA, układ Intel® Cyclone® 10 LP posiada wewnętrzny oscylator 50MHz. Port GPIF układu FX3 może działać z maksymalnym taktowaniem 100MHz, z tego powodu został zastosowany układ PLL działający z taką częstotliwością, jak zaprezentowano na rysunku 3.3. Dla szerokości słowa 32 bity i taktowaniu 100MHz otrzymujemy maksymalną prędkość transmisji 3200Mbps.



Rys. 3.3. Generacja zegara

Przy ustawnionym taktowaniu 100MHz, w czasie transmisji okazało się jednak że występują błędy transmisji, co prawdopodobnie spowodowane jest wykorzystanym adapterem. W związku z tym zmniejszono taktowanie do 50MHz rozwiązuając problem błędów transmisji. Spowodowało to ograniczenie teoretyczne prędkości transmisji do 1600Mbps, lecz wykorzystanie innego adaptera powinno pozwolić uruchomić transmisję przy 100MHz.

3.2. Aplikacja hosta

Aplikacja działająca w przestrzeni hosta połączonego z kartą pomiarową została stworzona celem obsługi transmisji danych, ich wizualizacji oraz konfiguracji urządzenia. Aplikacja została napisana w języku C++ w wieloplatformowym środowisku QT. Do komunikacji ze sprzętem wykorzystywana jest biblioteka Cypress CyApi dostarczana przez producenta kontrolera USB.

3.2.1. Interfejs graficzny

Jednym z najważniejszych zadań stawianych przed interfejsem graficznym jest wyświetlenie przebiegów zebranych przez kartę pomiarową oraz wygodny ich podgląd poprzez możliwość przybliżania/oddalania przebiegów oraz ich przesuwania. W tym celu została wykorzystana biblioteka QCustomPlot.



Rys. 3.4. GUI aplikacji

Każdy kanał jest przedstawiany jako osobny graf na wykresie, wymaga to wyodrębnienia poszczególnych bitów z tablicy otrzymanych danych. Ze względu na to że grafy wymagają podania danych jako pary punktów **double x, double y** równomiernie rozłożenie zostało osiągnięte poprzez odpowiednie zwiększenie offsetu dla współrzędnej **y** przy każdym kolejnym kanale. Zostało to przedstawione w listingu 3.1. Łączenie punktów w ciągły wykres zostało osiągnięte poprzez skonfigurowanie dla wszystkich grafów stylu linii jako **lsStepLeft**, przez co

wartość lewej punktu jest utrzymywana aż do prawego punktu, powtarzając operację dla każdej pary punktów.

```
QVector<double> *x_vect = new QVector<double>;
QVector<double> *y_vect = new QVector<double>;
x_vect->reserve(new_data.size());
y_vect->reserve(new_data.size());

for (int j=0; j<new_data.size(); ++j)
{
    x_vect->insert(j, j);
}

for (unsigned int i=0; i<8; ++i)
{
    for (int j=0; j<new_data.size(); ++j)
    {
        y_vect->insert(j, ((new_data.at(j) >> i) & 1)*1.25 + i*2.5);
    }

    ui->qplot->graph(7-i)->setData(*x_vect, *y_vect, TRUE);

    y_vect->clear();
}

delete x_vect;
delete y_vect;
```

Listing 3.1. Parsowanie danych do rysowania przebiegów

Pozostałe funkcjonalności interfejsu GUI to:

- Wybór urządzenia do połączenia
- Wybór wybór endpointów IN/OUT dla danego urządzenia
- Kontrola wyjść GPIO
- Wybór źródła dla danego kanału
- Czyszczenie wykresu
- Rozpoczęcie akwizycji danych z wybraną ilością transakcji

3.2.2. Obsługa USB API

Biblioteka Cypress CyAPI wykorzystywana jest do komunikacji z układem Cypress FX3. Odpowiada ona za:

- Enumeracje urządzeń
- Enumeracje endpointów urządzenia
- Wysyłanie danych
- Odbieranie danych

Listing 3.2 przedstawia kod wykorzystywany do wysyłania danych do karty pomiarowej. Do funkcji przekazywana jest lista, która następnie jest kopowana do buforu o rozmiarze jednego transferu i wysyłana. Rozmiar transferu skonfigurowany jest jako 16 pakietów po 1024 bajty, co wynika z konfiguracji endpointów jako obsługujące tryb burst, pozwalając na przesłanie 16 pakietów w jednym transferze.

```

LONG packet_length = TX_TRANSFER_SIZE;
UCHAR outBuf[TX_TRANSFER_SIZE] = {0};

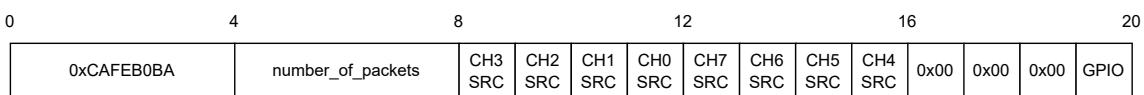
for (int i=0; i<tx_buf.size(); i++)
{
    outBuf[i] = tx_buf.at(i);
}

BOOL status = BulkOutEpt->XferData(outBuf, packet_length);

```

Listing 3.2. Wysyłanie transferu

Instrukcje konfiguracyjne wysyłane do karty pomiarowej mają format jak przedstawiony na rysunku 3.5 i składają się z 20 bajtów. Jako pierwsza wysyłana jest preambuła ramki, następnie podawana jest ilość pakietów które powinny zostać odesłane przez kartę. Kolejne 8 bajtów stanowi konfigurację źródła danych dla poszczególnych kanałów. Na koniec podawane jest jaki stan wyjść GPIO powinien być ustalony w FPGA. Dane są przesyłane w formacie little-endian.



Rys. 3.5. Przesyłana ramka danych

W przypadku odbierania danych przykładowy kod został przedstawiony w listingu 3.3. Wykonywany asynchronicznie składa się on z 3 części, w przeciwieństwie do synchronicznego XferData użytego w listingu 3.2. Mimo bardziej skomplikowanej obsługi pozwala kolejkować zapytania o dane jeszcze zanim wyślemy instrukcje do urządzenia w celu jak najszybszej obsługi otrzymywanych danych. Dodatkowo według dokumentacji biblioteki CyAPI[19] kolejkowanie wielu zapytań pozwala zwiększyć wydajność aplikacji.

```
CCyUSBDevice *USBDevice = new CCyUSBDevice(NULL);
OVERLAPPED inOvLap;
inOvLap.hEvent = CreateEvent(NULL, false, false, L"CYUSB_IN");

unsigned char inBuf[128];
LONG length = 128;

UCHAR *inContext = USBDevice->BulkInEndPt->BeginDataXfer(inBuf, length, &inOvLap);

USBDevice->BulkInEndPt->WaitForXfer(&inOvLap, 100);

USBDevice->BulkInEndPt->FinishDataXfer(inBuf, length, &inOvLap, inContext);

CloseHandle(inOvLap.hEvent);
```

Listing 3.3. Przykładowy kod do odbierania danych

Listing 3.4 przedstawia pseudokod faktycznego rozwiązania wykorzystanego do przy odbieraniu danych. W nawiasach “/* */” umieszczono opis zastępujący właściwy kod. Na początku przygotowywane są bufore, aby pozwolić na obsłużenie wszystkich transferów bez potrzebny dodatkowego kopiowania danych w trakcie odbierania danych. Dodatkowo została stworzona kolejka zapytań do odbierania danych, aby przyspieszyć obsługę transferów USB.

Testy wykazały, że zmiana rozmiaru kolejki nie ma dużego wpływu na prędkości uzyskiwane w trakcie testów w rozdziale 4. Prawdopodobnie jest to związane z faktem że interfejs FIFO działa z połową prędkości co sprawia że użycie kolejki nie jest konieczne, dopóki nie zostanie przywrócona pełna prędkość transmisji.

```

int buffer_counter = 0;

for (unsigned int i=0; i < transfer_count; i++)
{
    /* Prepare buffer for every transfer */
}

for (unsigned int i=0; i < QUEUE_SIZE && i < transfer_count; i++)
{
    /* Queue up transactions for better performance, use buffer pointed by (buffer_counter) */
    buffer_counter++; // Count buffers used by queue
}

this->send_bulk(tx_buf); // Send request for data

int q_ctr = 0; // Queue counter
int read_offset = buffer_counter; // Number of buffers already used in queue

while (transfer_ctr) // For every transfer
{
    /* Wait for transfer pointed by (q_ctr) */

    /* Transfer pointed by (q_ctr) finished */
    /* Read data into buffer (buffer_counter - read_offset) */

    if (buffer_counter < transfer_count) // If not all buffers used
    {
        /* Add new request in queue pointed by (q_ctr) to replace finished transfer request,
        use buffer pointed by (buffer_counter) */
        buffer_counter++;
    }

    q_ctr++; // Move to next request in queue
    transfer_ctr--; // 1 transfer finished
    if (q_ctr == QUEUE_SIZE) // Loop the queue
    {
        q_ctr = 0;
    }
}

/* Copy all buffers into list */
/* Free allocated memory */

```

Listing 3.4. Pseudokod odbierania danych z kolejką

W przypadku enumeracji jak przedstawiono w listingu 3.5 wystarczy sprawdzić ilość dostępnych urządzeń USB, a następnie spróbować je otworzyć i zapisać do listy.

```
CCyUSBDevice *USBDevice;
USBDevice = new CCyUSBDevice(NULL, CYUSBDRV_GUID, true);

for (int nCount = 0; nCount < USBDevice->DeviceCount(); nCount++)
{
    USBDevice->Open(nCount);
    /* Save the device */
    USBDevice->Close();
}
delete USBDevice;
```

Listing 3.5. Enumeracja urządzeń

Gdy sprawdzamy dostępne endpointy dla danego urządzenia wymaga to nieco więcej trudu, jak widzimy w listingu 3.6. Najpierw należy otworzyć wcześniej wybrane urządzenie, po czym iterując po wszystkich interfejsach oferowanych przez to urządzenie możemy sprawdzić dostępne endpointy. W naszym przypadku interesują nas tylko Endpointy BULK, które dzielimy później na IN i OUT.

```
this->selectedDevice->Open(nDeviceIndex);
int interfaces = this->selectedDevice->AltIntfcCount() + 1; /* AltIntfcCount - find-out how many
   ↳ alternate interfaces are exposed by the device */

for (int nDeviceInterfaces = 0; nDeviceInterfaces < interfaces; nDeviceInterfaces++)
{
    this->selectedDevice->SetAltIntfc(nDeviceInterfaces);
    int eptCnt = this->selectedDevice->EndPointCount();

    for (int endPoint = 1; endPoint < eptCnt; endPoint++)
    {
        CCyUSBEndPoint *ept = this->selectedDevice->EndPoints[endPoint];

        if (ept->Attributes == 2) /* Endpoint BULK */
        {
            if (ept->bIn) /* Save the endpoint IN */
            else /* Save the endpoint OUT */
        }
    }
}
```

Listing 3.6. Enumeracja endpointów

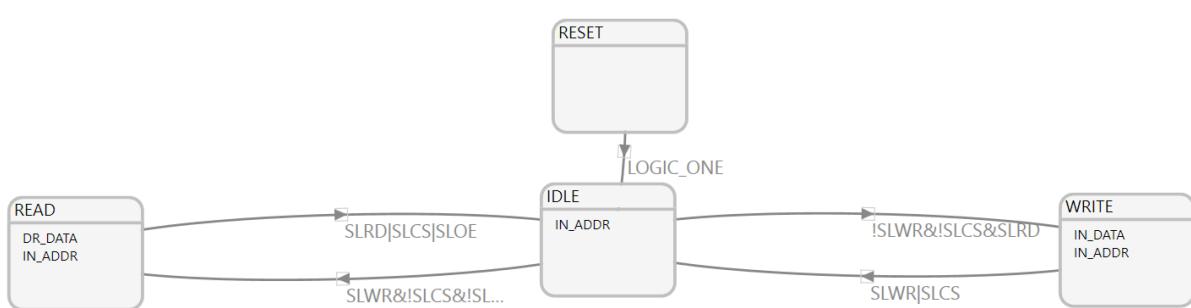
3.3. Kontroler USB 3.0

Cypress FX3 jest kontrolerem USB 3.0 posiadającym wiele funkcjonalności. FX3 oferuje wsparcie dla interfejsów USB 2.0 i USB 3.0, posiada interfejs GPIF 2, jest to skrót od General Programmable Interface Generation 2, oraz takie peryferia jak UART, SPI, I2C, I2S. Dodatkowo dzięki rozbudowanemu systemowi DMA, wiele operacji może się wykonywać bez udziału procesora.

3.3.1. GPIF

Interfejs o nazwie General Programmable Interface II, jest częścią Processor Interface Block (PIB) stanowiącego interfejs do zewnętrznego procesora lub układu FPGA o wysokiej przepustowości. Najważniejsze funkcje bloku GPIF[2]:

- Możliwość działania jako master lub slave.
- Posiada 256 programowalnych stanów.
- Wspiera równoległą magistralę danych o szerokościach 8/16/24/32.
- Wspierane 14 konfigurowalnych pinów gdy używana jest magistrala 32 bitowa, w innym wypadku 16 pinów.
- Piny kontrolne mogą być wejściami/wyjściami lub dwukierunkowe.
- Wewnętrzne liczniki i komparatory.
- Szerokie możliwości konfiguracji maszyny stanów.



Rys. 3.6. gpif_sm

Rysunek 3.6 przedstawia maszynę stanów skonfigurowaną w porcie GPIF. Jak wcześniej pokazano na rysunku 3.2 port GPIF jest skonfigurowany jako synchroniczne slave Fifo, używające zewnętrzny sygnał zegarowy i szerokości magistrali danych 32 bity.

Oprócz przejściowego stanu RESET, maszyna stanów posiada 3 podstawowe stany.

IDLE – Port GPIF nie wysyła, ani nie odbiera danych. Sprawdzana jest magistrala adresowa, aby na bieżąco aktualizować wybrany wątek.

READ – Port GPIF w dalszym ciągu sprawdza aktualny adres, ale dodatkowo steruje magistralą wysyłając nią słowa danych do mastera z każdym cyklem zegara.

WRITE – Podobnie jak w stanie READ, sprawdzamy adres, lecz w tym wypadku czytamy dane otrzymywane magistralą od zewnętrznego układu. Z każdym cyklem zegara dane są zapisywane w FIFO.

Taka konfiguracja daje nam 4 możliwe przejścia pomiędzy stanami, należy też pamiętać, że stanem aktywnym sygnałów jest stan niski.

IDLE - READ – Aby rozpocząć wysyłanie danych sygnały !SLCS,!SLRD,!SLOE muszą być w stanie aktywnym, natomiast SLWR powinien być nieaktywny.

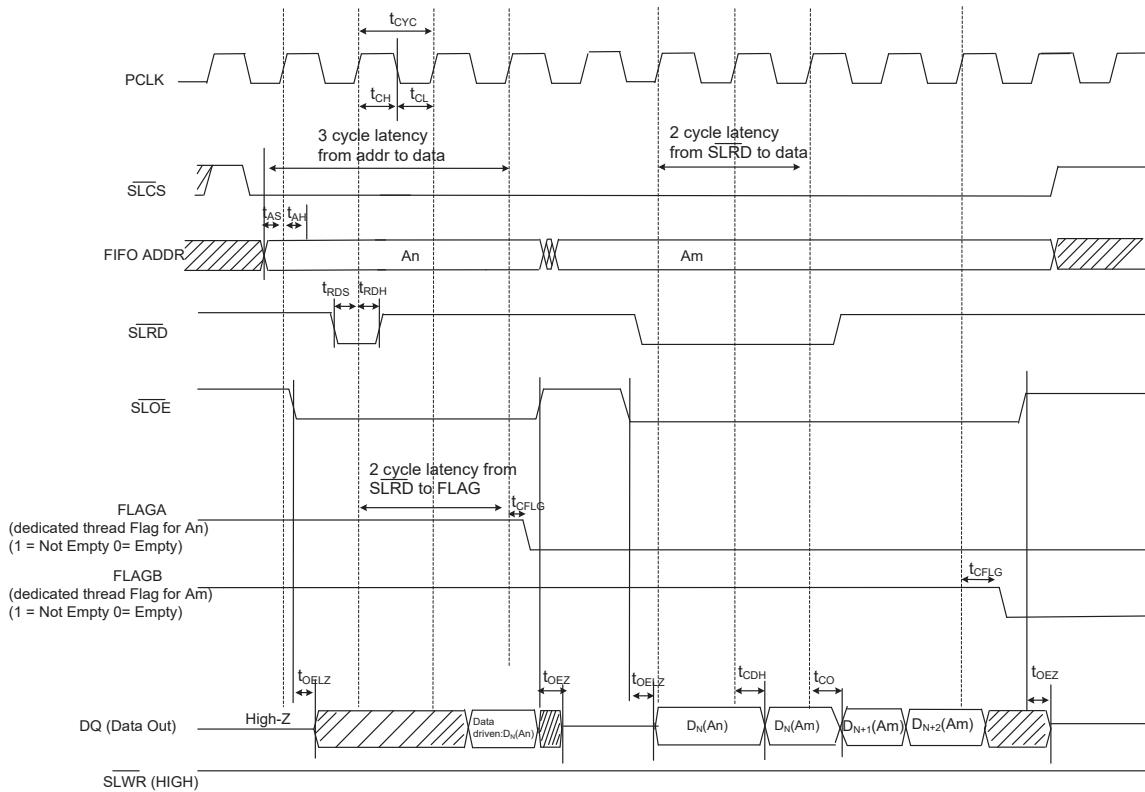
READ - IDLE – Którykolwiek z sygnałów SLRD,SLCS,SCOE przestanie być w stanie aktywnym.

IDLE - WRITE – Aby rozpoczęć odbieranie danych sygnały !SLCS,!SLWR muszą być w stanie aktywnym, natomiast SLRD powinien być nieaktywny.

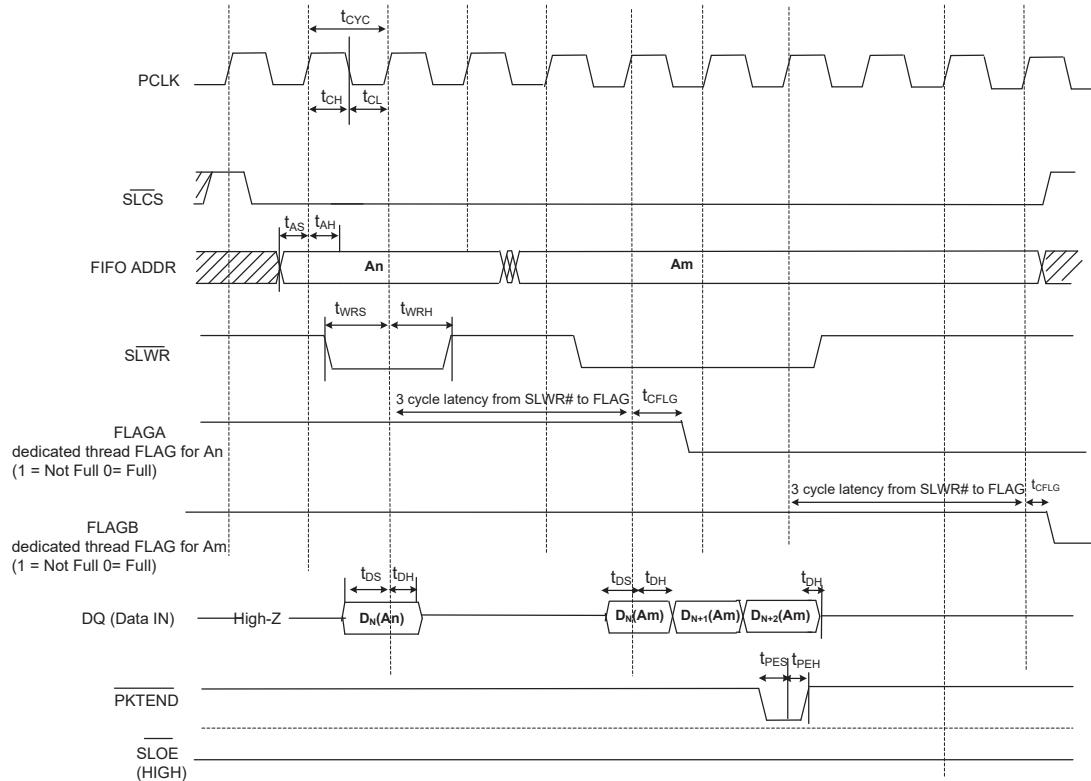
WRITE - IDLE – Którykolwiek z sygnałów SLWR,SLCS przestanie być w stanie aktywnym.

Rysunki 3.8 i 3.7 przedstawiają ogólną sekwencję zapisu i odczytu podaną przez producenta. Dokładna implementacja obsługi FIFO jest opisana w sekcji 3.4.1.

W naszym przypadku zawsze korzystamy z zapisów i odczytów w trybie burst, gdyż pojedyncze operacje byłyby nieoptymalne. Dodatkowo linia SLCS, jest zawsze ustawiona w stanie aktywny, jako że do FIFO połączony jest tylko jeden układ oraz sygnał PKEND jest ustawiony w tryb nieaktywny, gdyż zawsze wysyłamy pełne pakiety danych.



Rys. 3.7. Sekwencja odczytu z FIFO w trybie burst[2]



Rys. 3.8. Sekwencja zapisu z FIFO w trybie burst[2]

Odczyt z FIFO dzieli się na następujące etapy.

1. Master odczytał flagę oznaczającą że FIFO jest gotowe do odczytu.
2. Master ustawia odpowiedni adres oraz pin SLCS.
3. Po ustabilizowaniu adresu SLOE jest ustawiany przez mastera w stan aktywny, pozwala-
- jąc FX3 sterować magistralą danych.
4. Master ustawia SLRD aby odczytać dane, które po 2 cyklach są ustawiane na magistralę.
5. Z każdym cyklem zegara PCLK FIFO aktualizuje swój stan i wystawia kolejne słowo
- danych na wyjścia.

Zapis do FIFO dzieli się na następujące etapy.

1. Master sprawdza flagę, czy możliwy jest zapis do FIFO
2. Master ustawia odpowiedni adres oraz pin SLCS.
3. Master ustawia SLWR w stan aktywny i podaje dane na magistralę.
4. Z każdym cyklem zegara dane są wpisywane do FIFO, poki SLWR jest w stanie aktywnym.

3.3.2. USB

Cypress FX3 pozwala skonfigurować 16 endpointów IN oraz 16 endpointów OUT w trybach: CONTROL, BULK, INTERRUPT, ISOCHRONOUS. Przepustowość USB w trybie 3.0 wynosi 5Gbps, natomiast w przypadku USB 2.0 może to być 480Mbps/12Mbps/1.5Mbps, w zależności możliwości portu USB.

W przypadku konfiguracji USB większość konfiguracji jest gotowa i należy jedynie skonfigurować endpointy w zależności od naszych potrzeb. W listingu 3.7 definiujemy 2 endpointy IN i OUT, które będą naszym konsumentem i producentem oraz ustawiamy długość nurst na 16, aby umożliwić USB przesyłanie pakietów 16 razy dłuższych niż standardowe tj. 1024 bajty.

```
#define BURST_LEN          (16)
#define CY_FX_EP_PRODUCER   0x01    /* EP 1 OUT */
#define CY_FX_EP_CONSUMER    0x81    /* EP 1 IN */
```

Listing 3.7. Parametry endpointów

Następnie nasze parametry wpisujemy w odpowiednie miejsce do deskryptorów jak pokazano w listingu 3.8, aby host do którego się połączymy mógł poprawnie rozpoznać nasze urządzenie. Ustawione parametry należy również ustawić dla samych endpointów przy uruchamianiu urządzenia przy użyciu funkcji CyU3PSetEpConfig, która pozwala ustawić opcje m.in. takie jak:

- Typ endpointu(Bulk, Interrupt, ..)
- Długość pakietów burst(1, 16, ..)
- Rozmiar pakietu(512, 1024, ..)

```

/* Endpoint descriptor for producer EP */
0x07, /* Descriptor size */
CY_U3P_USB_ENDPNT_DESCR, /* Endpoint descriptor type */
CY_FX_EP_PRODUCER, /* Endpoint address and description */
CY_U3P_USB_EP_BULK, /* Bulk endpoint type */
0x00, 0x04, /* Max packet size = 1024 bytes */
0x00, /* Servicing interval for data transfers : 0 for bulk */

/* Super speed endpoint companion descriptor for producer EP */
0x06, /* Descriptor size */
CY_U3P_SS_EP_COMPN_DESCR, /* SS endpoint companion descriptor type */
BURST_LEN-1, /* Max no. of packets in a burst : 0: burst 1 packet at a time */
0x00, /* Max streams for bulk EP = 0 (No streams) */
0x00, 0x00, /* Service interval for the EP : 0 for bulk */

/* Endpoint descriptor for consumer EP */
0x07, /* Descriptor size */
CY_U3P_USB_ENDPNT_DESCR, /* Endpoint descriptor type */
CY_FX_EP_CONSUMER, /* Endpoint address and description */
CY_U3P_USB_EP_BULK, /* Bulk endpoint type */
0x00, 0x04, /* Max packet size = 1024 bytes */
0x00, /* Servicing interval for data transfers : 0 for Bulk */

/* Super speed endpoint companion descriptor for consumer EP */
0x06, /* Descriptor size */
CY_U3P_SS_EP_COMPN_DESCR, /* SS endpoint companion descriptor type */
BURST_LEN-1, /* Max no. of packets in a burst : 0: burst 1 packet at a time */
0x00, /* Max streams for bulk EP = 0 (No streams) */
0x00, 0x00 /* Service interval for the EP : 0 for bulk */

```

Listing 3.8. Deskryptor endpointów

3.3.3. DMA

Aby zrozumieć jak działają transfery w układzie FX3 oraz w jaki sposób można wykorzystać DMA, należy wytłumaczyć kilka pojęć.

Socket – Stanowi punkt połączenia między blokami sprzętowymi oraz pamięcią RAM. Do socket'u może być przypisane deskryptor DMA oraz flagi właściwe dla tego socket'u.

Deskryptor DMA – Instrukcje dla DMA przechowywane w rejestrach pamięci w RAM. Przechowują informację na temat adresu i rozmiaru bufora oraz wskaźnik na następny deskryptor.

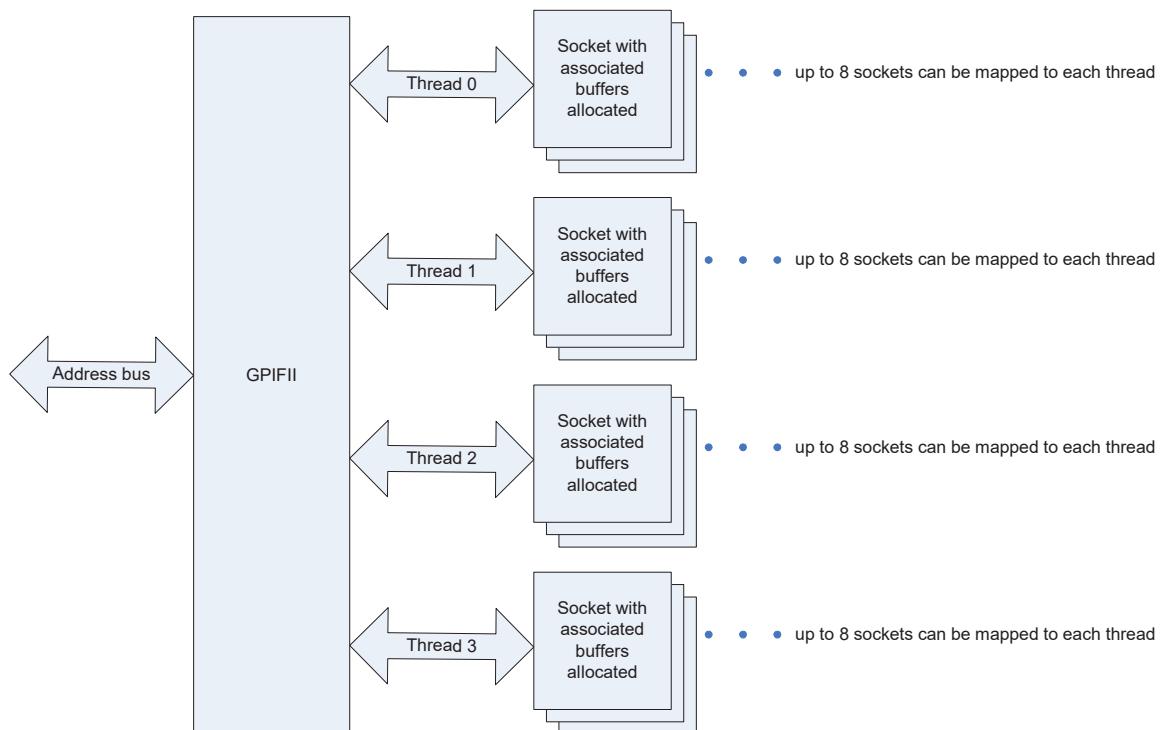
Bufor DMA – Obszar w pamięci RAM służący do przechowywania danych przesyłanych przez układ.

Wątek GPIF – Stanowi połączenie między interfejsem GPIF, a socket'em umożliwiając przepływ danych przy odczycie i zapisie.

Aktualna konfiguracja wątków GPIF

Wątek 3 – Wykorzystywany do wysyłania danych z hosta do karty pomiarowej

Wątki 0 i 1 – Wykorzystywane do przesyłania danych z karty pomiarowej do hosta.



Rys. 3.9. Wątki i odpowiadające im 32 socket'y[2]

Jak przedstawiono na rysunku 3.9 wątki są częścią bloku PIB stanowiąc połączenie pomiędzy GPIF, a socket'ami z przypisanymi do nich buforami DMA. FX3 posiada 32 socket'y dostępne dla PIB, w aktualnym trybie, gdy używamy 2 bitowej magistrali adresowej wszystkie 4 wątki są używane, natomiast socket'y są do nich przypisane w formie modulo 4. W przypadku wykorzystania większej liczby linii adresowych używany jest tylko jeden wątek, który z pomocą dodatkowej logiki potrafi obsługiwać swobodnie pozostałe socket'y, jednak prowadzi to do wprowadzenia dodatkowych opóźnień.

W przypadku socket'ów przeznaczonych do transmisji USB przewidziane jest 16 socket'ów producentów i 16 konsumentów, które są na stałe przypisane do odpowiadających im numerami endpointów.

Aby skonfigurować przesył danych od host do karty pomiarowej została użyta konfiguracja została przedstawiona w listingu 3.9. Analizując możemy wyszczególnić:

- Alokujemy 16KB pamięci na bufor DMA, wynikające z ustawienia parametru burst na 16 oraz rozmiarowi standardowego pakietu USB 1024 bajty.
- Ilość buforów DMA ustawiamy na 1 z racji że do karty wysyłane są jedynie dane konfiguracyjne które nie przekraczają rozmiaru jednego bufora.
- Ustawiamy DMA w tryb zliczania bajtów, ponieważ rozmiar pakietów w naszym przypadku się nie zmienia.
- Ustawiamy jako producenta socket CY_U3P_UIB_SOCKET_PROD_1 będący endpoint'em 0x01
- Ustawiamy konsumenta na CY_U3P_PIB_SOCKET_3, czyli wykorzystujemy wątek 3 GPIF, który wykorzystywany jest do wysyłania danych na port GPIF.
- Tworzymy DMA w trybie AUTO, gdyż manualne akceptowanie transakcji nie jest potrzebne gdy nie modyfikujemy przesyłanych danych oraz zwiększy to wydajność.

```

/* Create a DMA channel for U2P transfer.
 * DMA size is set based on the USB speed. */
dmaCfg.size = DMA_BUF_SIZE * size; // 16*1024
dmaCfg.count = CY_FX_SLFIFO_DMA_BUF_COUNT_U_2_P; // 1
dmaCfg.dmaMode = CY_U3P_DMA_MODE_BYTE;
dmaCfg.prodSckId = CY_U3P_UIB_SOCKET_PROD_1; // Endpoint 0x01
dmaCfg.consSckId = CY_U3P_PIB_SOCKET_3; // Thread 3

apiRetStatus = CyU3PDmaChannelCreate(&glChHandleSl_fifoUtoP, CY_U3P_DMA_TYPE_AUTO, &dmaCfg);

```

Listing 3.9. Konfiguracja DMA do wysyłania danych do karty

W przypadku przesyłania danych z karty pomiarowej do USB wykorzystana została konfiguracja Multi Channel DMA pozwalająca na przesyłanie danych z kilku socketów do jednego. W przypadku konfiguracji możemy wyszczególnić

- Ponownie alokujemy 16KB pamięci na bufor DMA i ustawiamy tryb zliczania bajtów.
- Ustawienie 2 socket'ów producentów i przypisanie im po 4 bufory celem zwiększenia wydajności.
- Ustawiamy jako konsumenta socket CY_U3P_UIB_SOCKET_CONS_1 będący endpoint'em 0x81
- Ustawiamy jako producenta socket'y CY_U3P_PIB_SOCKET_0 oraz CY_U3P_PIB_SOCKET_1, wykorzystując wątki 0 oraz 1 do odczytu danych z GPIF.
- Tworzymy DMA w trybie AUTO_MANY_TO_ONE, aby dane były przesyłane z obu producentów do jednego konsumenta.

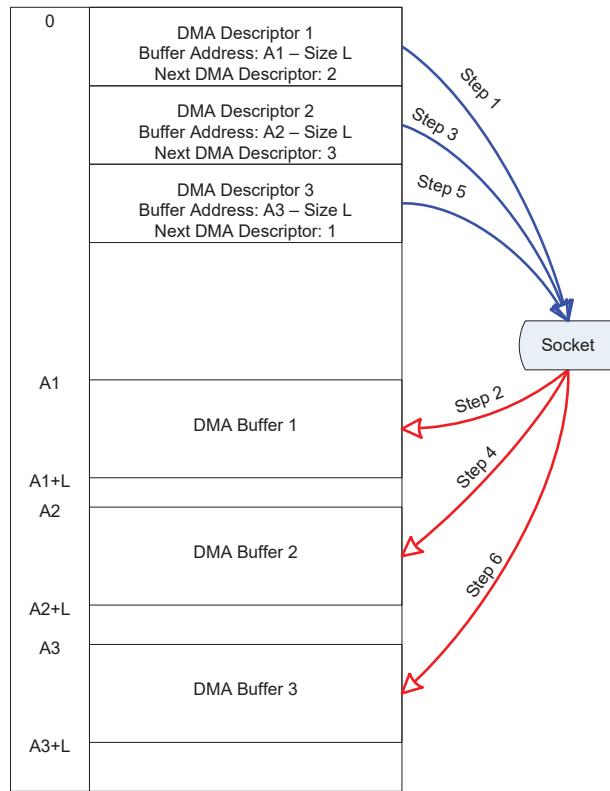
```
/* Create a Multi DMA channel for P2U transfer. */
dmaMultiCfg.size = DMA_BUF_SIZE * size; // 16+1024
dmaMultiCfg.count = CY_FX_SLFIFO_DMA_BUF_COUNT_P_2_U; // 4
dmaMultiCfg.validSckCount = 2;
dmaMultiCfg.prodSckId[0] = CY_U3P_PIB_SOCKET_0; // Thread 0
dmaMultiCfg.prodSckId[1] = CY_U3P_PIB_SOCKET_1; // Thread 1
dmaMultiCfg.consSckId[0] = CY_U3P_UIB_SOCKET_CONS_1;
dmaMultiCfg.dmaMode = CY_U3P_DMA_MODE_BYTE;

apiRetStatus = CyU3PDmaMultiChannelCreate(&glChHandleSlFifoPtoU,
→ CY_U3P_DMA_TYPE_AUTO_MANY_TO_ONE, &dmaMultiCfg);
```

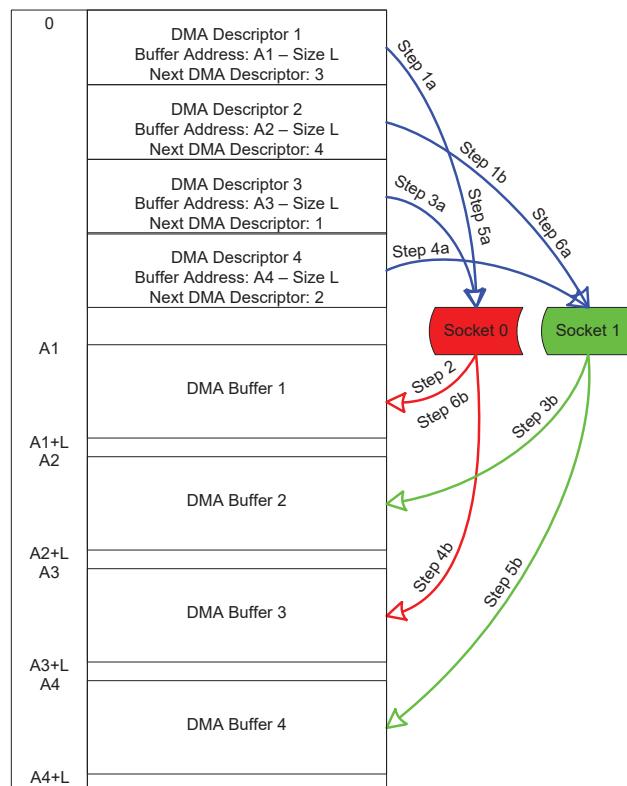
Listing 3.10. Konfiguracja MultiDMA do odbierania danych z karty

Wykorzystanie trybu Multi Channel DMA było konieczne, aby umożliwić ciągłe przesyłanie danych z karty pomiarowej bez stosowania wewnętrznego bufora w FPGA. W standardowym przypadku jak przedstawiono na rysunku 3.10 do odczytu danych wykorzystywany jeden wątek współpracujący z jednym socket'em.

W trakcie gdy bufor DMA się zapełni, urządzenie podłączone do portu GPIF przerywa wysyłanie danych po odczytaniu flag sygnalizujących zapełnienie buforu. Powoduje to konieczność implementacji kontroli przepływu danych po stronie zewnętrznego urządzenia oraz zmniejszenie przepustowości poprzez przestoje w transmisji danych, gdyż zmiana bufora trwa około $1\mu S$.

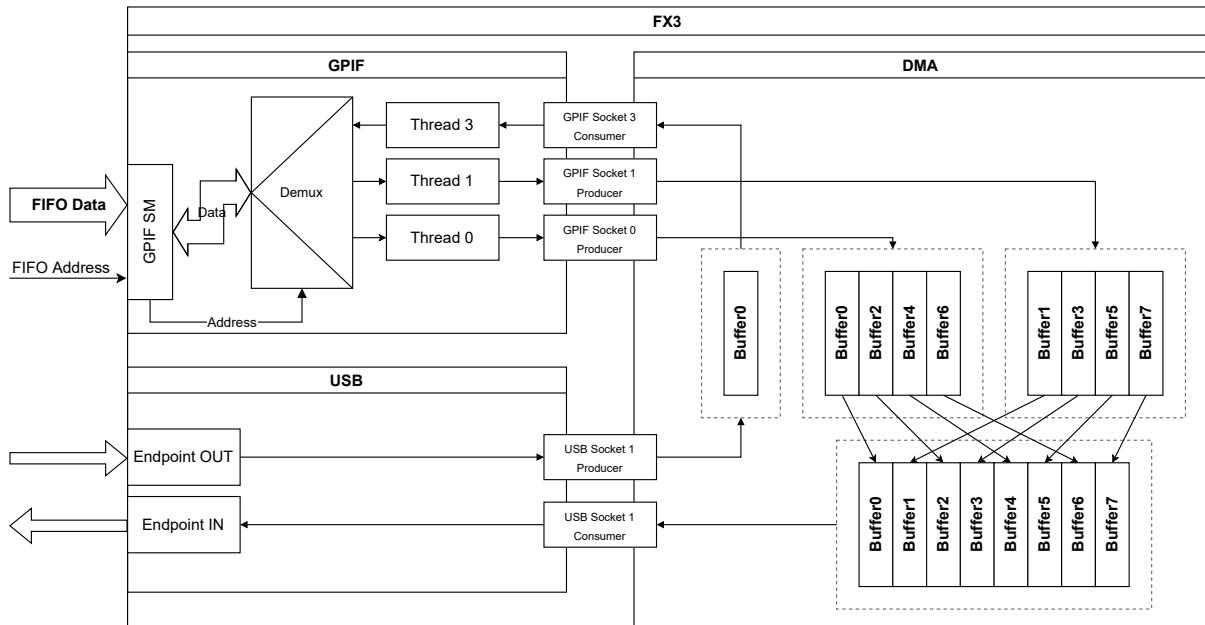


Rys. 3.10. Wykorzystanie jednego socket'u do transmisji[20]



Rys. 3.11. Wykorzystanie dwóch socket'ów do transmisji[20]

Alternatywnym podejściem jest zastosowanie 2 socketów, do których dane zapisywane są na przemian. Z racji że przełączenie wątku trwa 1 cykl zegara, możliwa jest zmiana wątku i kontynuowanie zapisu danych do innego socket'u w trakcie trwania transmisji. Podejście to zostało zaprezentowane na rysunku 3.11. Oba wątki naprzemiennie pobierają bufore umożliwiając ciągłą transmisję do buforu endpoint'u USB.



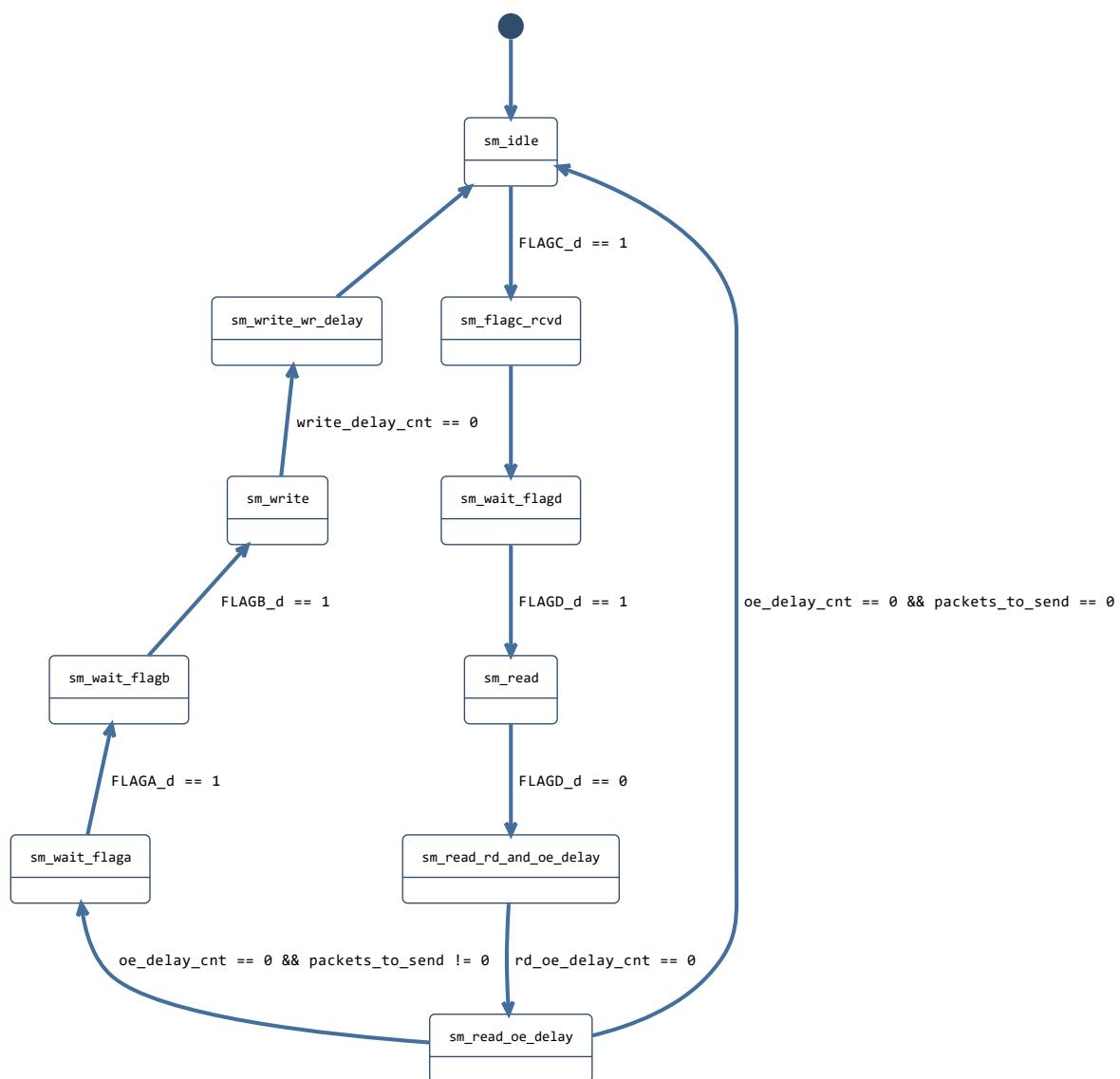
Rys. 3.12. Przesyłanie danych DMA

Rysunek 3.12 przedstawia faktyczną konfigurację zastosowaną w przypadku opisywanej karty pomiarowej. Układ FPGA steruje linią adresową, wybierając wątek 3 w trakcie odbierania danych oraz naprzemiennie wątki 0 i 1 w trakcie nadawania, których bufore są później łączone za pomocą DMA przed wysłaniem do endpoint'u USB.

3.4. Układ FPGA

3.4.1. Interfejs FIFO

Na rysunku 3.13 przedstawiona jest maszyna stanów służąca do obsługi interfejsu Slave Fifo w układzie FPGA. Operacja układu zawsze rozpoczyna się od oczekiwania na pakiet, po odebraniu którego jest on dekodowany i sprawdzane jest ile pakietów powinno być odesłane. W razie gdy odpowiedź nie jest wymagana maszyna stanów wraca do stanu IDLE po odebraniu pakietu.



Rys. 3.13. Maszyna stanów obsługująca interfejs FIFO

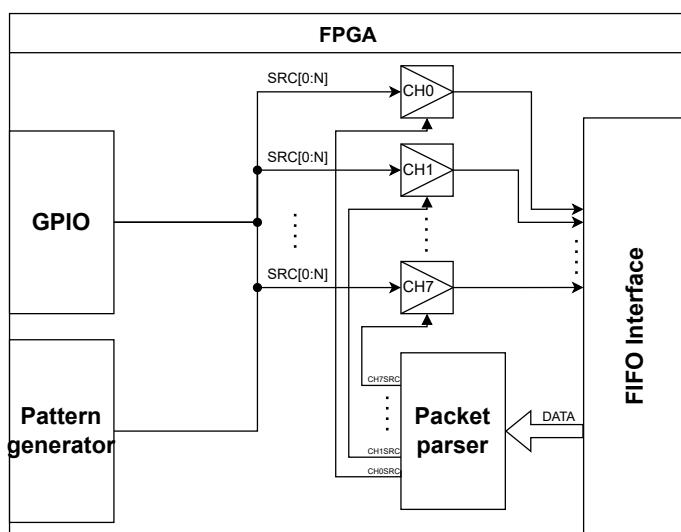
W tabeli 3.4 został umieszczony opis stanów wraz z oczekiwanyimi stanami sygnałów w interfejsie FIFO.

Tabela 3.4. Opis poszczególnych stanów

Nazwa stanu	Stan wyjść
sm_idle	SLOE=1 SLRD=1 SLWR=1 ADDR=0 Oczekujemy na ustawienie flagi C, wtedy DMA będzie gotowe do wysyłania danych
sm_flagc_rcvd	SLOE=1 SLRD=1 SLWR=1 ADDR=3 Gdy DMA jest gotowe wchodzimy do tego stanu na 1 cykl zegara.
sm_wait_flagd	SLOE=1 SLRD=1 SLWR=1 ADDR=3 Oczekujemy na ustawienie flagi D, co oznacza że w FIFO są dane do odebrania.
sm_read	SLOE=0 SLRD=0 SLWR=1 ADDR=3 W tym stanie FPGA odczytuje dane otrzymywane z FIFO.
sm_read_rd_and_oe_delay	SLOE=0 SLRD=0 SLWR=1 ADDR=3 Zapewnienia że SLRD będzie ustawione przez 3 cykle zegara po ustawieniu flagi D na 0.
sm_read_oe_delay	SLOE=0 SLRD=1 SLWR=1 ADDR=3 Zapewnienia że SLOE będzie ustawione przez 5 cykli zegara po ustawieniu flagi D na 0. Jeśli ilość żądanego transakcji jest równa zero, przechodzimy do stanu IDLE.
sm_wait_flaga	SLOE=1 SLRD=1 SLWR=1 ADDR=1 Oczekujemy na ustawienie flagi A, wtedy DMA będzie gotowe do odbierania danych.
sm_wait_flagb	SLOE=1 SLRD=1 SLWR=1 ADDR=1 Oczekujemy na ustawienie flagi D, co oznacza że w FIFO jest puste i można wysyłać dane.
sm_write	SLOE=1 SLRD=1 SLWR=0 ADDR=0/1 Dane są wpisywane do FIFO, dopóki nie zostanie wysłana żądana liczba 32-bitowych słów, tj. <code>packets_to_send * 16384/4</code> . Co 4096 słów następuje zmiana adresu aby kontynuować ciągły transfer.
sm_write_wr_delay	SLOE=1 SLRD=1 SLWR=1 ADDR=0 Zapewnia wymagane utrzymanie SLWR=1 przez jeden cykl po zakończeniu zapisu.

3.4.2. Obsługa kanałów

Ze względu na dużą ilość dostępnych pinów oraz możliwość wewnętrznego generowania sygnałów, została dodana możliwość wyboru źródła z którego pobierane są dane i przekazywane do wybranego kanału. Rysunek 3.12 przedstawia zaimplementowaną logikę. Pakiet otrzymany od hosta jest parsowany, a następnie przekazywany do multiplekserów na wejściu każdego kanału. Do każdego multipleksera doprowadzone są wszystkie dostępne sygnały z których następnie wybierany jest jeden.



Rys. 3.14. Wybór kanałów

Aby umożliwić weryfikację przesyłanych danych został dodany 8-bitowy licznik licznik taktowany zegarem 4 razy szybszym niż część logiki obsługująca FIFO.

```
// data generator counter
always @(posedge clk_pll*4, negedge reset_)begin
    if(!reset_)begin
        data_gen <= 8'd0;
    end else begin
        data_gen <= data_gen + 1;
    end
end
```

Listing 3.11. Licznik służący do przeprowadzania testów

Implementacja wyboru kanału i próbkowania danych została przedstawiona w listingu 3.12.

data_src – stanowi połączenie sygnałów od **przycisków, stałego zera** oraz **wewnętrznego 8-bitowego licznika**,

ch_src – zawiera wybrany kanał parsowany z otrzymanej ramki danych.

Aby móc zwiększyć częstotliwość próbkowania 4-krotnie, zmniejszono ilość kanałów z 32 do 8. Aktualnie przy interfejsie FIFO działającym z częstotliwością 50MHz, dla próbkowania wejść oraz licznika wynosi ona 200MHz. Z każdym cyklem zegara do rejestru **ch_data_d** dopisywane jest 8 bitów, co znaczy że po 4 cyklach zegara otrzymamy pełne 32 bitowe słowo, które następnie zostanie wysłane interfejsem FIFO.

Aby uniknąć metastabilności przy zmianie domen zegara oraz przy próbkowaniu pinów GPIO, zastosowano prosty synchronizator wykonany z 2 rejestrów[21].

```

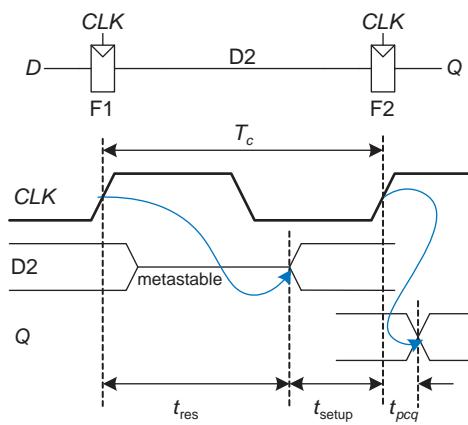
assign data_src = {PB_2d, 1'b0, data_gen};

// Invert bits order to match app
assign ch_data[7] = data_src[ch_src[0]]; // CH0
assign ch_data[6] = data_src[ch_src[1]]; // CH1
assign ch_data[5] = data_src[ch_src[2]]; // CH2
assign ch_data[4] = data_src[ch_src[3]]; // CH3
assign ch_data[3] = data_src[ch_src[4]]; // CH4
assign ch_data[2] = data_src[ch_src[5]]; // CH5
assign ch_data[1] = data_src[ch_src[6]]; // CH6
assign ch_data[0] = data_src[ch_src[7]]; // CH7

// Channels data shifter
always @ (posedge clk_pll4, negedge reset_) begin
    if (!reset_) begin
        ch_data_d <= 32'd0;
    end else begin
        ch_data_d <= {ch_data, ch_data_d[31:8]};
    end
end

```

Listing 3.12. Wybieranie źródła dla kanału



Rys. 3.15. Synchronizator zbudowany z 2 rejestrów[21]

3.4.3. Parsowanie pakietów

W celu parsowania pakietów został dodany licznik, który będzie wskazywał numer odbieranego pakietu. Z racji że dane są odbiera 2 cykle po ustawnieniu sygnału **SLRD**, wykorzystujemy w tym celu opóźniony sygnał **SLRD_d3_** i gdy jest w stanie niskim zaczynamy liczenie pakietów. Dodatkowo sprawdzana jest preambuła pakietu, która stanowi pierwszy odebrany pakiet, jej weryfikacja pozwala sprawdzić czy mamy do czynienia z faktycznym pakietem, czy innymi nieprzewidzianymi danymi.

```

assign read_ready = (SLRD_d3_ == 1'b0);
assign read_data = (SLRD_d3_ == 1'b0) ? DQ_d : 32'd0;

// Read packet index
always @(posedge clk_pll, negedge reset_) begin
    if (!reset_) begin
        packet_index <= 0;
    end else if (read_ready) begin
        packet_index <= packet_index + 1;
    end else begin
        packet_index <= 0;
    end
end

// Check header
always @(posedge clk_pll, negedge reset_) begin
    if (!reset_) begin
        valid_packet <= 0;
    end else if (!read_ready) begin
        valid_packet <= 0;
    end else if (read_ready && packet_index == 0 && read_data == 32'hCAFEB0BA) begin
        valid_packet <= 1;
    end
end

```

Listing 3.13. Liczenie pakietów i sprawdzanie preambuły

0	4	8	12	16	20								
0xCAFEB0BA	number_of_packets	CH3 SRC	CH2 SRC	CH1 SRC	CH0 SRC	CH7 SRC	CH6 SRC	CH5 SRC	CH4 SRC	0x00	0x00	0x00	GPIO

Rys. 3.16. Przesyłana ramka danych

Gdy już posiadamy indeks aktualnego pakietu oraz mamy pewność że pakiet jest prawidłowy możemy w prosty sposób odczytać dane jak przedstawiono na rysunku 3.14.

```
// Parse packet
always @(posedge clk_pll, negedge reset_) begin
    if (!reset_) begin
        packets_to_send <= 32'h0;
        ch_src[0] <= 8'h8;
        ch_src[1] <= 8'h8;
        ch_src[2] <= 8'h8;
        ch_src[3] <= 8'h8;
        ch_src[4] <= 8'h8;
        ch_src[5] <= 8'h8;
        ch_src[6] <= 8'h8;
        ch_src[7] <= 8'h8;
        USER_LED <= 4'b1111;
    end else if (valid_packet) begin
        case (packet_index)
            1: begin
                packets_to_send <= read_data;
            end
            2: begin
                ch_src[0] <= read_data[7:0];
                ch_src[1] <= read_data[15:8];
                ch_src[2] <= read_data[23:16];
                ch_src[3] <= read_data[31:24];
            end
            3: begin
                ch_src[4] <= read_data[7:0];
                ch_src[5] <= read_data[15:8];
                ch_src[6] <= read_data[23:16];
                ch_src[7] <= read_data[31:24];
            end
            4: begin
                USER_LED <= ~read_data[3:0];
            end
        endcase
    end
end
```

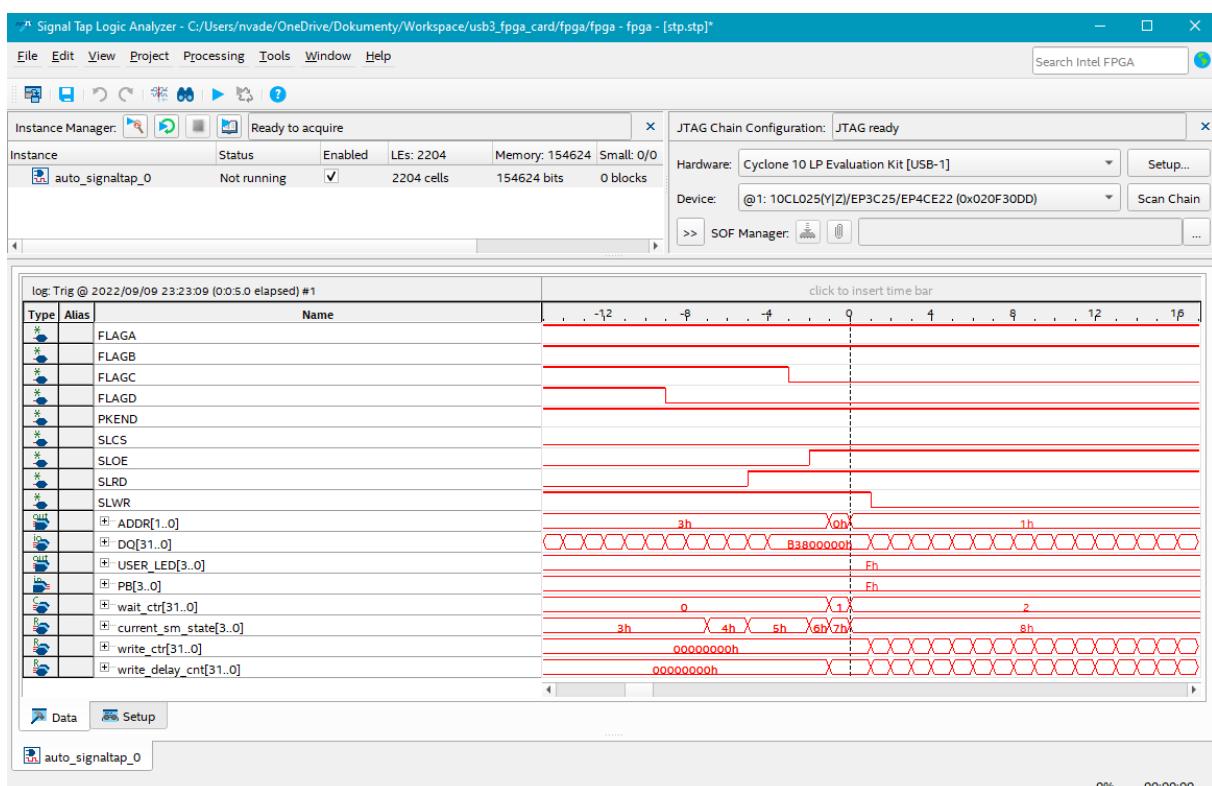
Listing 3.14. Parsowanie otrzymanego pakietu przez FPGA

3.4.4. Informacje debugowe

W trakcie dewelopowania kodu bardzo przydatne okazały się 2 narzędzia.

Pierwszym z nich jest Signal Tap Logic Analyzer, którego interfejs przedstawiono na rysunku 3.17. Pozwala on na próbkowanie w czasie rzeczywistym dowolnie wybranych sygnałów, a następnie ich wyświetlenie. Dodatkowo dzięki możliwości ustawienia trigger'ów dla wybranych sygnałów, pozwoliło to debugować kluczowe momenty transmisji interfejsem Slave FIFO.

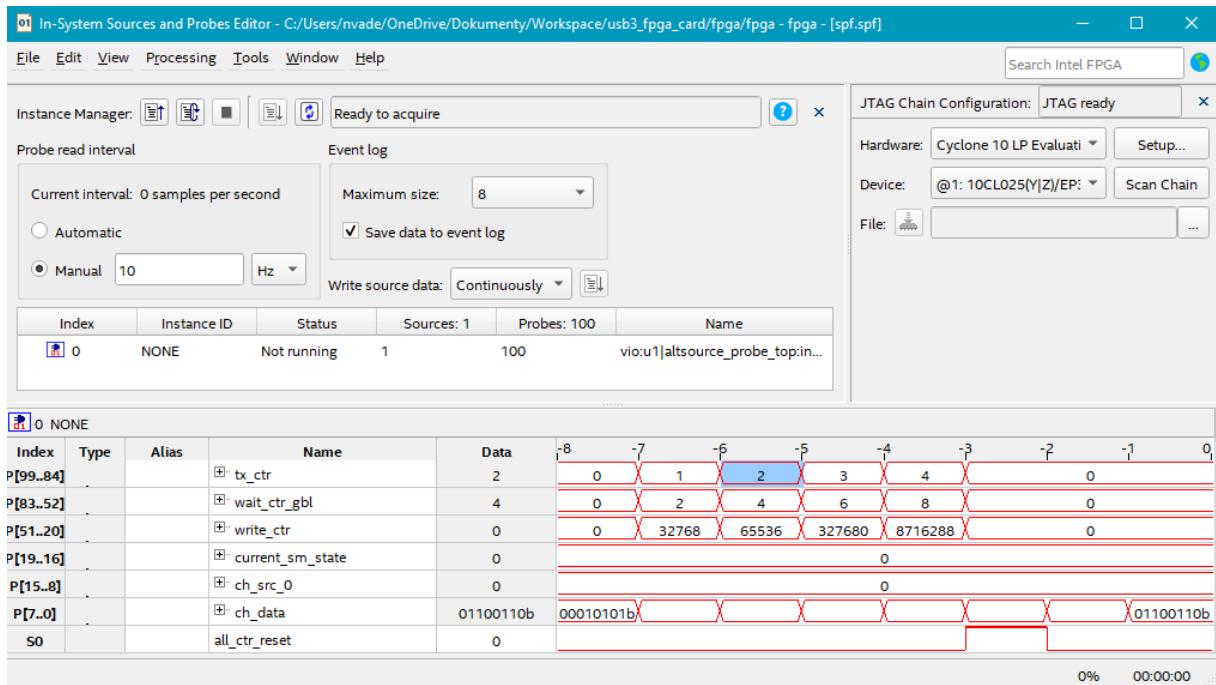
Drugim z programów jest In-System Sources and Probes, przedstawiony na rysunku 3.18. Pozwalał on w wygodny sposób na odczytanie wartości dodanego portu wejściowego jednym kliknięciem oraz sterowanie portem wyjściowym. W przeciwieństwie do analizatora Signal Tap, konieczne było dodanie instancji modułu do projektu.



Rys. 3.17. Interfejs Signal Tap Logic Analyzer

```
vio ul (
    .probe({tx_ctr, wait_ctr_gbl, write_ctr, current_sm_state, ch_src[0], ch_data}),
    .source(all_ctr_reset)
);
```

Listing 3.15. Instancja In-System Sources and Probes

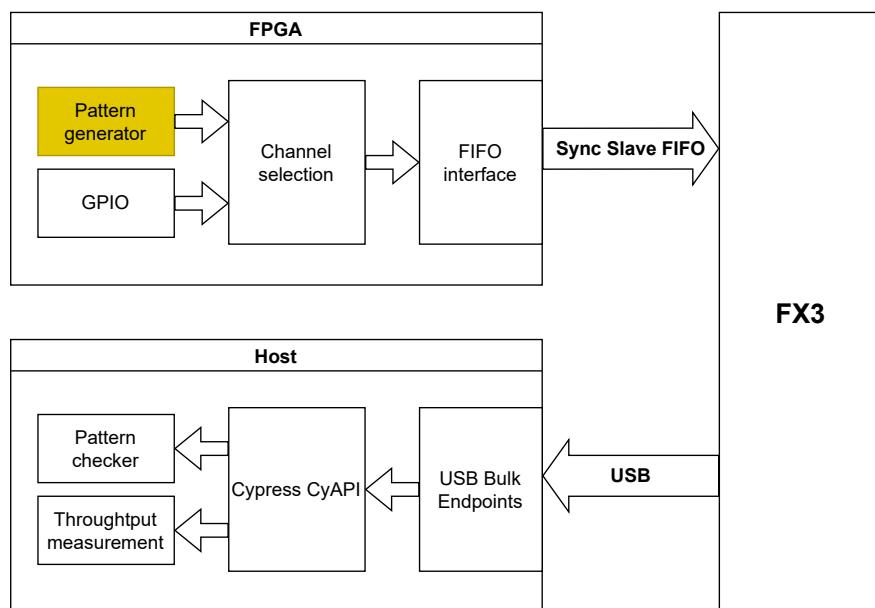


4. Weryfikacja rozwiązania

Celem weryfikacji poprawności komunikacji między układowej jak i jej przepustowości zostały wykonane 2 rodzaje testów zgodnie z procedurą opisaną poniżej.

4.1. Procedura testowa

W trakcie testowania sprzęt był podpięty w standardowej konfiguracji tj. FPGA połączone do portu GPIF układu FX3 za pomocą adaptera, całość z kolei podpięta do hosta z systemem Windows i uruchomioną aplikacją stworzoną w ramach pracy. Na rysunku 4.1 przedstawiony jest diagram przepływu danych.



Rys. 4.1. Przepływ danych testowych

Dane były przesyłane z bloku **Pattern Generator**, funkcjonującym jako licznik inkrementowany o 1 z każdym cyklem zegara. Kanały w aplikacji hosta były skonfigurowane, aby odczytywać wszystkie bity licznika, jak przedstawiono w tabeli 4.1. Bez wykorzystania zewnętrznego generatora podpiętego do pinów GPIO układu FPGA, nie można było wykonać pełnych testów

E2E. Jednak jako że najważniejsze była weryfikacja protokołu komunikacji, było to wystarczające podejście.

Tabela 4.1. Konfiguracja kanałów

Kanał	Źródło
CH0	fpga_counter0
CH1	fpga_counter1
CH2	fpga_counter2
CH3	fpga_counter3
CH4	fpga_counter4
CH5	fpga_counter5
CH6	fpga_counter6
CH7	fpga_counter7

W trakcie testów próby zostały wykonane kolejno dla różnych ilości żądanych transferów: 1,8,16,64,128,256 Każdy transfer składał się z 8 pakietów po 16KB każdy, dla długości “burst” ustawionej na 16.

4.2. Poprawność danych

Weryfikacja danych została dodana jako kod znajdujący się pod dyrektywą `#ifdef DEBUG_PATTERN`, która jest domyślnie wyłączona.

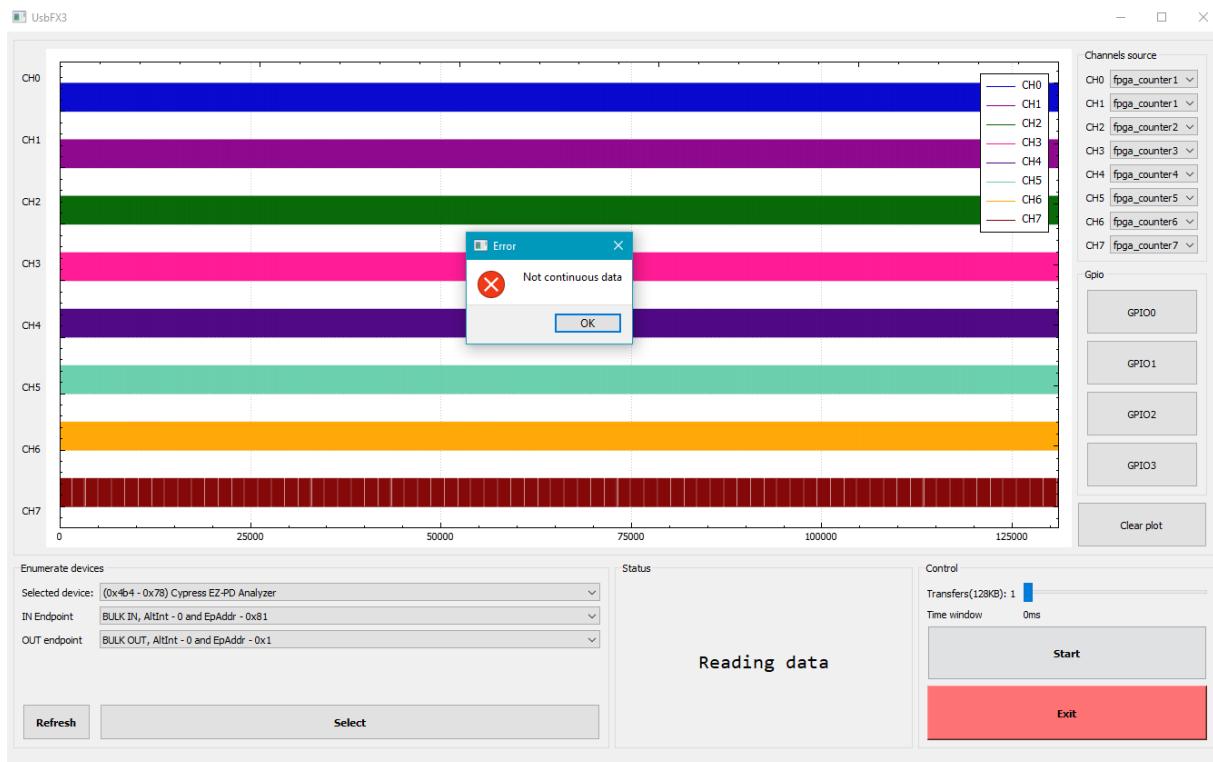
```
QMessageBox messageBox;
int old_byte = -1;
int new_byte;
for (int i = 0; i < this->data_buffer.size(); i++)
{
    new_byte = reverse(this->data_buffer.at(i)); /* Correct endianness */
    if ((new_byte - old_byte) != 1 && new_byte != 0 && old_byte != 255 && old_byte != -1)
    {
        qDebug() << "Not continuous data new:" << new_byte << "old:" << old_byte;
        messageBox.critical(0, "Error", "Not continuous data");
        messageBox.setFixedSize(500, 200);
        break;
    }
    old_byte = new_byte;
}
```

Listing 4.1. Weryfikacja danych

Jak przedstawiono w listingu 4.1, po otrzymaniu danych i przeniesieniu ich do listy sprawdzane jest czy wszystkie kolejne wartości zwiększą się o 1. W przeciwnym wypadku na ekranie wyświetla się komunikat z błędem. Oczywiście obsługiwane są również przypadki gdy wartość licznika zmieni się z 255 na 0 oraz gdy sprawdzamy pierwszy bajt.

Testy wykazały brak błędów dla przypadków testowych **1,8,16,64,128,256** transferów.

Aby pokazać działanie mechanizmu zmieniono konfigurację CH0 z fpga_counter0 na fpga_counter1, co spowodowało nieciągłość danych jak przedstawiono na rysunku 4.2.



Rys. 4.2. Przykładowy błąd danych

4.3. Przepustowość

W przypadku przepustowości w teorii nie jest potrzebne wykonywanie żadnych testów, z racji że wykorzystanie **DMA Dual Socket** pozwala na nieprzerwany przesył danych z FPGA. Dla ustawionego zegara 50MHz dla Slave Fifo w układzie FX3 i szerokości magistrali 32 bit, otrzymujemy wartość teoretyczną 1600Mbps.

Aby jednak sprawdzić czy w praktyce nasz układ działa tak samo został zmierzony czas trwania odbioru danych. W trakcie testów okazało się że w zależności od tego, czy pomiar czasu był zaczęty przed, czy po wysłaniu request'u do FPGA, otrzymywaliśmy różne wartości szczególnie, dla małej liczby transakcji.

```
QEapsedTimer timer;
qint64 elapsed_time;
...
// Prepare buffers/API for read data.
// Send request to read data.
...
timer.start();
...
// Receive requested packets
...
elapsed_time = timer.nsecsElapsed();
...
// Gather data from buffers into rx_buf
...
qDebug() << "Elapsed time:" << elapsed_time/1000 << "us";
qDebug() << "Estimated speed:" << (qint64)rx_buf.size()*8*1000/elapsed_time << "Mbps";
```

Listing 4.2. Pomiar przepustowości, uruchomienie timera po wysłaniu request'u

```
QEapsedTimer timer;
qint64 elapsed_time;
...
// Prepare buffers/API for read data.
...
timer.start();
...
// Send request to read data.
...
// Receive requested packets
...
elapsed_time = timer.nsecsElapsed();
...
// Gather data from buffers into rx_buf
...
qDebug() << "Elapsed time:" << elapsed_time/1000 << "us";
qDebug() << "Estimated speed:" << (qint64)rx_buf.size()*8*1000/elapsed_time << "Mbps";
```

Listing 4.3. Pomiar przepustowości, uruchomienie timera przed wysłaniem request'u

Listingi 4.2 i 4.3 przedstawiają kod użyty do pomiaru czasu i obliczenia prędkości przesyłu. Domyślnie kod odpowiedzialny za pomiar czasu jest wyłączony, ale można go włączyć dodając define dla DEBUG_TIME lub DEBUG_TIME_V2, w zależności od tego która wersja pomiaru ma być użyta.

Biorąc pod uwagę fakt, że zaraz po wysłaniu request'u rozpoczęty jest odbiór danych, w przypadku uruchomienia timera pod fakcie, uzyskamy wartości jak w tabeli 4.2, bardzo

Tabela 4.2. Uzyskane prędkości dla pomiaru z uruchomieniem timera po wysłaniu request'u

Ilość transferów	Prędkość
1	4378Mbps
8	1737Mbps
16	1668Mbps
64	1615Mbps
128	1608Mbps
256	1604Mbps

Tabela 4.3. Uzyskane prędkości dla pomiaru z uruchomieniem timera przed wysłaniem request'u

Ilość transferów	Prędkość
1	1041Mbps
8	1494Mbps
16	1542Mbps
64	1586Mbps
128	1591Mbps
256	1596Mbps

zawyżone dla jednej transakcji. Odwrotna sytuacja występuje w tabeli 4.3 gdy pomiar czasu zaczniemy przed wysłaniem request'u, wtedy czas wysłania pakietu dodaje się do czasu odbierania, obniżając wyniki.

Aby złagodzić ten problem najlepiej wykonać wiele transferów, co zniweluje błąd pomiaru. Finalnie w obu przypadkach wartości zbiegają się w kierunku wartości 1600Mbps co potwierdza wcześniejsze obliczenia.

5. Podsumowanie i wnioski

Celem przedstawionej pracy było stworzenie cyfrowej karty pomiarowej z interfejsem USB 3.0. Wszystkie zaplanowane kroki udało się zrealizować.

Zbudowany system składa się z kontrolera USB, układu FPGA oraz aplikacji na hosta.

- Kontroler USB enumeruje się jako urządzenie z 2 endpoint'ami typu BULK oraz udostępnia interfejs FIFO dla układu FPGA.
- Układ FPGA próbkuję dane wejściowe 8 kanałów z częstotliwością 200MHz oraz udostępnia sygnały z wewnętrznego licznika użytego do weryfikacji rozwiązania. Możliwy jest swobodny wybór źródła sygnału dla danego kanału. Próbkowane dane są wysyłane interfejsem FIFO do kontrolera USB. Układ dodatkowo pozwala sterować diodami znajdującymi się na płytce używając parametru w otrzymywanej ramce danych.
- Aplikacja wyświetla dane, pozwala przybliżać oraz przesuwać wykres. Dane odbierane są bez przerw i transmisja działa poprawnie nawet przy odbiorze dużej ilości danych. Aplikacja posiada interfejs do kontroli kanałów, GPIO oraz ilości odbieranych transferów. Odbierane dane mogą być weryfikowane oraz mierzona jest uzyskana przepustowość.

W ramach rozwoju projektu warto przetestować działanie kontrolera USB z innym adapterem i uruchomić interfejs FIFO z pełną prędkością, co pozwoli zwiększyć przepustowość dwukrotnie. Dodatkowo warto pomyśleć nad opcją integracji stworzonego rozwiązania z otwartoźródłowym projektem **sigrok**, który umożliwiłby jego współpracę z oprogramowaniem do analizy sygnałów. Pozwoliłoby to korzystać z trigger'ów oraz używać narzędzi dekodujących uchwycone transmisje.

Wykaz skrótów

E2E	End-to-end
FPGA	Field-programmable gate array
GPIF	General Programmable Interface
IP Core	Intellectual property core
PCIe	Peripheral Component Interconnect Express
PIB	Processor Interface Block
UART	Universal asynchronous receiver-transmitter
USB	Universal Serial Bus

Bibliografia

- [1] Jan Axelson. *USB Complete The Developer's Guide, Fifth Edition*. Lakeview Research LLC, 2015.
- [2] *EZ-USB® FX3™ Technical Reference Manual*. Rev. *F. Infineon. 2019.
- [3] *USB*. Wikipedia. URL: <https://en.wikipedia.org/wiki/USB>.
- [4] *Saleae Logic Pro 16*. Sigrok. URL: https://sigrok.org/wiki/Saleae_Logic_Pro_16.
- [5] *DreamSourceLab DSLogic U3Pro16*. Sigrok. URL: https://sigrok.org/wiki/DreamSourceLab_DSLogic_U3Pro16.
- [6] *Spartan-6 Family Overview*. v2.0. Xilinx. 2011.
- [7] *High Speed SelectIO Wizard*. v3.6. Xilinx. 2020.
- [8] *Saleae Logic16*. Sigrok. URL: https://sigrok.org/wiki/Saleae_Logic16.
- [9] *DreamSourceLab DSLogic Plus*. Sigrok. URL: https://sigrok.org/wiki/DreamSourceLab_DSLogic_Pro.
- [10] *Kingst LA5016*. Sigrok. URL: https://sigrok.org/wiki/Kingst_LA_Series.
- [11] *LeCroy LogicStudio*. Sigrok. URL: https://sigrok.org/wiki/LeCroy_LogicStudio.
- [12] *Saleae Logic*. Sigrok. URL: https://sigrok.org/wiki/Saleae_Logic.
- [13] *ASIX OMEGA*. Sigrok. URL: https://sigrok.org/wiki/ASIX_OMEGA.
- [14] *7 Series FPGAs Integrated Block for PCI Express*. v3.3. Xilinx. 2020.
- [15] *LiteScope*. EnjoyDigital. URL: <https://github.com/enjoy-digital/litescope>.
- [16] *Cypress EZ-USB FX2LP-based Logic Analyzer using Open Source sigrok PulseView*. Infineon. URL: <https://community.infineon.com/t5/Knowledge-Base-Articles/Cypress-EZ-USB-FX2LP-based-Logic-Analyzer-using-Open-Source-sigrok-PulseView/ta-p/252866>.

- [17] *EZ-USB™ FX3 Explorer kit as 16-channel 100 MHz logic analyzer with sigrok PulseView*. Infineon. URL: <https://community.infineon.com/t5/Knowledge-Base-Articles/EZ-USB-FX3-Explorer-kit-as-16-channel-100-MHz-logic-analyzer-with-sigrok/ta-p/283993>.
- [18] *FT232H*. Adafruit. URL: <https://www.adafruit.com/product/2264>.
- [19] *Cypress CyAPI Programmer's Reference*. Cypress Semiconductor. 2018.
- [20] *Getting Started with EZ-USB® FX3*. Rev. *J. Infineon. 2021.
- [21] Sarah L. Harris David Money Harris. *Digital Design and Computer Architecture*. Denise E. M. Penrose, 2007.