Uniwersytet Mikołaja Kopernika  
Wydział Fizyki, Astronomii i Informatyki Stosowanej

Mariusz Wiśniewski  
254019

Praca inżynierska  
na kierunku Automatyka i Robotyka

Interfejs szybkiej wymiany danych pomiędzy układem FPGA a kontrolerem CYUSB3014 z wykorzystaniem programowalnej maszyny stanów GPIF II

Opiekun pracy dyplomowej  
dr Robert Frankowski  
Zakład Fizyki Technicznej i Zastosowań Fizyki

Toruń 2015

|  |  |
| --- | --- |
| Pracę przyjmuję i akceptuję | Potwierdzam złożenie pracy dyplomowej |
|  |  |
| ....................................................... | ....................................................... |
| *data i podpis opiekuna pracy* | *data i podpis dziekanatu* |

Serdeczne dziękuję  
dr Robertowi Frankowskiemu  
za pomoc i wsparcie merytoryczne.

*UMK zastrzega sobie prawo własności niniejszej pracy inżynierskiej w celu udostępniania dla potrzeb działalności naukowo-badawczej lub dydaktycznej*

**Streszczenie**

W niniejszej pracy inżynierskiej przedstawiono treści związane z opracowaniem interfejsu wymiany danych pomiędzy komputerem osobistym, a układem programowalnym FPGA z wykorzystaniem zestawu EZ-USB FX3 firmy Cypress.

Omówiono zagadnienia teoretyczne związane z magistralą USB, interfejsem synchronicznej kolejki FIFO, flagami wskazującymi dostępność gniazd FX3 oraz programowalną maszyną stanów GPIF II. Opisano sposób realizacji projektu prezentując krok po kroku konfigurację GPIF II, omawiając maszyny o skończonej liczbie stanów zaimplementowane w układzie programowalnym odpowiadające za strumieniowy transfer danych. Wykorzystano do tego celu również moduły przygotowane przez firmę Xilinx. Programy dla kontrolera USB konfigurujące gniazda, punkty końcowe magistrali USB oraz kanały DMA (bezpośredniego dostępu do pamięci) oparto na źródłach dostarczonych przez producenta, czyli firmę Cypress.

Praca składa się z 9 rozdziałów, z których pierwszy stanowi wstęp do tematyki projektu wraz z określeniem celu pracy.

Rozdział drugi ma na celu przybliżyć standard USB, począwszy od USB 2.0 aż do najnowszej wersji, czyli 3.0. Omówiono w nim najistotniejsze elementy składające się na magistralę oraz poszczególne rodzaje transmisji danych, takie jak kontrolna, przerwaniowa, masowa i izochroniczna.

Rozdział trzeci skupia się na omówieniu interfejsu synchronicznej kolejki FIFO (ang. Synchronous Slave FIFO Interface) i programowalnej maszyny stanów GPIF II. Zaprezentowano najważniejsze sygnały oraz koncept użycia ich podczas transmisji. Na potrzeby pracy dyplomowej zaprojektowano PCB łączący oba układy. Opisano także krok po kroku konfigurację interfejsu używając dedykowanego oprogramowania firmy Cypress, czyli GPIF II Designer.

Rozdział czwarty, dotyczący zestawu EZ-USB FX3, został podzielony na dwie części: teoretyczną oraz praktyczną stronę realizacji interfejsu kolejki FIFO. Najpierw omówiono niezbędną teorię na temat mechanizmów i opcji wykorzystywanych przez kontroler FX3, takich jak gniazda, deskryptory DMA czy kanały DMA. Dużo uwagi poświecono prawidłowej konfiguracji flag pełnych oraz częściowych. Realizacja oprogramowania opierała się na dwóch projektach: strumieniowym transferze danych oraz transmisji danych w dwóch kierunkach.

Rozdział piąty opisuje program napisany od podstaw dla FPGA Spartan 3E. Kod został oparty na maszynach o skończonej liczbie stanów. Dla każdej transmisji danych zostały przedstawione diagramy i opisy zmiany stanów logicznych sygnałów zarządzających magistralą. Ponadto skupiono się na wykorzystaniu zaimplementowanych modułach przygotowanych przez firmę Xilinx, takich jak moduł Cyfrowego Zarządzania Sygnałem Zegarowym oraz pamięć FIFO.

W ostatnim rozdziale zamieszczono informacje potwierdzające bezbłędną transmisję danych (zrzuty ekranów programów USB Control Center oraz Cypress Streamer) oraz porównanie szybkości transmisji w zależności od liczby przesyłanych pakietów.

Spis treści

[**1. Wstęp** 7](#_Toc412462827)

[1.1. Cel pracy 7](#_Toc412462828)

[1.2. Lista wykorzystanego oprogramowania 8](#_Toc412462829)

[**2. Uniwersalna magistrala szeregowa** 9](#_Toc412462830)

[2.1. Wybrane zagadnienia teoretyczne 9](#_Toc412462831)

[2.2.1. Punkty końcowe oraz potoki 9](#_Toc412462832)

[2.2.2. Deskryptory 10](#_Toc412462833)

[2.2.3. Typy transferów USB 10](#_Toc412462834)

[2.2.4. Detekcja i enumeracja urządzenia 11](#_Toc412462835)

[2.2. Standard USB 2.0 11](#_Toc412462836)

[2.3. Standard USB 3.0 12](#_Toc412462837)

[**3. Interfejs synchronicznej kolejki FIFO oraz programowalna maszyna stanów GPIF II** 14](#_Toc412462838)

[3.1. Interfejs synchronicznej kolejki FIFO 14](#_Toc412462839)

[3.2. Projekt obwodu drukowanego łączącego dwa zestawy uruchomieniowe 15](#_Toc412462840)

[3.3. Programowalna maszyna stanów GPIF II 17](#_Toc412462841)

[3.4. Narzędzie GPIF II Designer 18](#_Toc412462842)

[3.4.1. Zdefiniowanie i konfiguracja sygnałów interfejsu 18](#_Toc412462843)

[3.4.2. Konfiguracja maszyny stanów 19](#_Toc412462844)

[3.4.3. Konfiguracja flag pełnych oraz częściowych 20](#_Toc412462845)

[3.4.4. Kompilacja projektu 21](#_Toc412462846)

[**4. Cypress EZ-USB FX3** 22](#_Toc412462847)

[4.1. Teoria FX3 - wątki, gniazda, DMA oraz deskryptory 24](#_Toc412462848)

[4.1.1. Gniazda 24](#_Toc412462849)

[4.1.2. Deskryptory DMA 24](#_Toc412462850)

[4.1.3. Bufor DMA 24](#_Toc412462851)

[4.1.4. Wątki GPIF II 25](#_Toc412462852)

[4.1.5. Konfiguracja kanału DMA 25](#_Toc412462853)

[4.2. Konfiguracja flag pełnych oraz częściowych 26](#_Toc412462854)

[4.2.1. Flagi dedykowane, związane z konkretnym wątkiem 26](#_Toc412462855)

[4.2.2. Flagi wskazujące na aktualny wątek 26](#_Toc412462856)

[4.2.3. Flagi częściowe 27](#_Toc412462857)

[4.3. Interfejs synchronicznej kolejki FIFO - realizacja oprogramowania dla FX3 27](#_Toc412462858)

[4.3.1. Zintegrowane środowisko programistyczne Cypress EZ-USB Suite 27](#_Toc412462859)

[4.3.2. Tryb jednokierunkowego wysyłania lub odbierania danych 28](#_Toc412462860)

[4.3.3. Tryb transmisji danych w obu kierunkach 30](#_Toc412462861)

[4.4. USB Control Center - programowanie układu 30](#_Toc412462862)

[**5. Xilinx Spartan 3E** 32](#_Toc412462863)

[5.1. Środowisko projektowe 32](#_Toc412462864)

[5.2. Realizacja projektu i przypisanie wyprowadzeń 32](#_Toc412462865)

[5.3. Moduł Cyfrowego Zarządzania Sygnałem Zegarowym (DCM - Digital Clock Manager) 34](#_Toc412462866)

[5.4. Pamięć FIFO 37](#_Toc412462867)

[5.5. Struktura programu dla FPGA 38](#_Toc412462868)

[5.5.1. Główny program - Slave Fifo Main 38](#_Toc412462869)

[5.5.2. Transmisja strumienia danych do komputera - Stream Write to FX3 40](#_Toc412462870)

[5.5.3. Odbiór strumienia danych z komputera - Stream Read from FX3 41](#_Toc412462871)

[5.5.4. Transmisja danych w obu kierunkach - Loopback Transfer 42](#_Toc412462872)

[5.6. Programowanie FPGA Spartan 3E 43](#_Toc412462873)

[5.6.1. Konfiguracja z wykorzystaniem interfejsu JTAG 43](#_Toc412462874)

[5.6.2. Konfiguracja z wykorzystaniem układu Xilinx XCF04S i pliku Flash PROM 44](#_Toc412462875)

[**6. Prezentacja wyników oraz efektów działania aplikacji** 46](#_Toc412462876)

[6.1. Transmisja strumienia danych z układu FPGA do komputera 46](#_Toc412462877)

[6.1.1. Test prędkości transferu danych 47](#_Toc412462878)

[6.1.2. Odczytanie pakietu danych w programie USB Control Center 47](#_Toc412462879)

[6.1.3. Zapianie pakietu danych do pliku 48](#_Toc412462880)

[6.2. Odbiór strumienia danych przez układ FPGA 48](#_Toc412462881)

[6.2.1. Test prędkości transferu danych 48](#_Toc412462882)

[6.2.2. Wysłanie pakietu danych w programie USB Control Center 49](#_Toc412462883)

[6.3. Transmisja danych w obu kierunkach 49](#_Toc412462884)

[6.4. Porównanie szybkości transferu danych w zależności od ilości przesyłanych pakietów 50](#_Toc412462885)

[**7. Podsumowanie** 53](#_Toc412462886)

[**8. Literatura** 54](#_Toc412462887)

[**9. Załącznik** 55](#_Toc412462888)

# 1. Wstęp

Uniwersalną magistralę szeregową (USB) opracowano w 1996 roku, a rok później zaczęto ją wdrażać do komputerów stacjonarnych na masową skalę [1]. W chwili obecnej większość komputerów osobistych obsługuje interfejs niezależnie od systemu operacyjnego. Ten standard odniósł spektakularny sukces, ponieważ w wielu przypadkach nie wymaga od użytkownika specjalistycznej wiedzy albo instalacji dedykowanych sterowników - urządzenia działają natychmiastowo po podłączeniu do komputera. Skorzystali na tym producenci urządzeń peryferyjnych produkujący pamięci przenośne FLASH, drukarki, myszki, klawiatury, aparaty fotograficzne oraz wiele innego sprzętu Plug and Play (podłącz i używaj).

Również rynek mikroprocesorów otworzył się na magistralę USB, czego dowodem są produkty firmy Atmel z serii Xmega [2], systemy wbudowane wykorzystujące układy oparte na rdzeniu ARM (komercyjny sukces odniosły płytki Raspberry Pi, Cubieboard) oraz przede wszystkim firmy produkujące inteligentne telefony komórkowe. Wyeliminowano stosowanie dodatkowych programatorów lub konwerterów USB w celu zaprogramowania procesora. Zmniejsza to poniesione koszty dla użytkowników końcowych. Obecnie USB wypiera zarówno w profesjonalnym, jak i w amatorskim zastosowaniu magistrale o niskiej przepustowości, na przykład popularny RS-232, I2C czy SPI.

## 1.1. Cel pracy

Celem pracy było opracowanie interfejsu wymiany danych pomiędzy układem programowalnym FPGA, a kontrolerem EZ-USB FX3 firmy Cypress z wykorzystaniem programowalnej maszyny stanów GPIF II.

Na potrzeby projektu przeanalizowano schematy elektryczne złącza FX2 w Spartanie 3E oraz GPIF II w FX3, aby móc zaprojektować obwód drukowany płytki łączącej oba zestawy w programie Altium Designer. Do montażu elementów SMD (konektor Samtec QTH-060-01-L-D-A) wykorzystano dedykowany statyw, podgrzewacz oraz stację lutowniczą ze strumieniem gorącego powietrza (ang. hot-air soldering).

Podczas projektowania interfejsu należało zapoznać się z obsługą programów: Xilinx ISE 14.1, Xilinx PlanAhead, Xilinx Impact, Cypress EZ USB Suite, Cypress GPIF II Designer, Cypress Streamer i Cypress USB Control Center. Jako pomoc przede wszystkim posłużyły opublikowane dokumentacje producentów oraz wiedza nabyta podczas zajęć praktycznych w czasie studiów.

Interfejs synchronicznej kolejki FIFO został od podstaw napisany dla układu Spartan 3E (XC3S500E) w języku VHDL. Do jego realizacji wykorzystano dwa bloki funkcjonalne dostarczone przez firmę Xilinx, a mianowicie blok Cyfrowego Zarządzania Sygnałem Zegarowym (DCM) oraz pamięć FIFO. Z kolei projekty strumieniowego transferu danych (wykorzystując przekaz masowy USB) dla FX3 zostały napisane w języku C. Konfigurację programowalnej maszyny stanów GPIF II przeprowadzono w programie GPIF II Designer.

Oprogramowanie dla FX3 musiało zawierać konfigurację m.in. GPIF II, kanałów DMA, gniazd oraz punktów końcowych. Do tego zadania niezbędna okazała się podstawowa wiedza na temat zasady działania magistrali USB, której poświęcono rozdział 2.

## 1.2. Lista wykorzystanego oprogramowania

Poniżej przedstawiono listę wykorzystanego w pracy dyplomowej oprogramowania wraz z opisem zadań (Tab. 1).

Tab. . Lista użytego oprogramowania wraz z opisem

|  |  |
| --- | --- |
| **Nazwa** | **Opis** |
| Altium Designer 14.3.9 | projekt obwodu drukowanego łączącego FPGA z FX3 |
| Xilinx Webpack ISE 14.1 | synteza i kompilacja projektu dla FGPA |
| Xilinx PlanAhead 14.1 | przypisanie wyprowadzeń do układu FPGA |
| Xilinx Impact 14.1 | programowanie FPGA z wykorzystaniem interfejsu JTAG |
| Cypress GPIF II Designer 1.0 | zdefiniowanie ustawień oraz maszyny stanów interfejsu GPIF II |
| Cypress EZ-USB Suite | edycja projektu i kompilacja dla EZ-USB FX3 |
| Cypress USB Control Center | programowanie FX3, odbiór i wysyłanie danych |
| Cypress Streamer | masowy transfer danych wykorzystujący USB |
| Git 1.9.4 | rozproszony system kontroli wersji projektów |

# 2. Uniwersalna magistrala szeregowa

Magistrala USB z uwagi na prostotę obsługi sprawdza się zarówno w projektach urządzeń produkowanych na skalę przemysłową, jak i w amatorskich konstrukcjach. Poniżej przedstawiono korzyści płynące z zastosowania omawianej magistrali.

Korzyści dla użytkowników końcowych [3]:

* łatwość obsługi - istnieje jeden standard dla wielu urządzeń, wytrzymałe konektory, automatyczne wykrycie i konfiguracja urządzenia w systemie, możliwość działania bez względu na system operacyjny, możliwość rozszerzenia portów USB przy użyciu HUB-ów, prostota instalacji sterowników,
* możliwość zasilania urządzeń napięciem 5V prosto z portu, brak zewnętrznych zasilaczy,
* stabilność transferu - USB nie jest wrażliwe jak magistrala RS-232 na środowisko zewnętrzne, w przypadku błędów możliwa jest retransmisja danych.

Korzyści dla projektantów urządzeń:

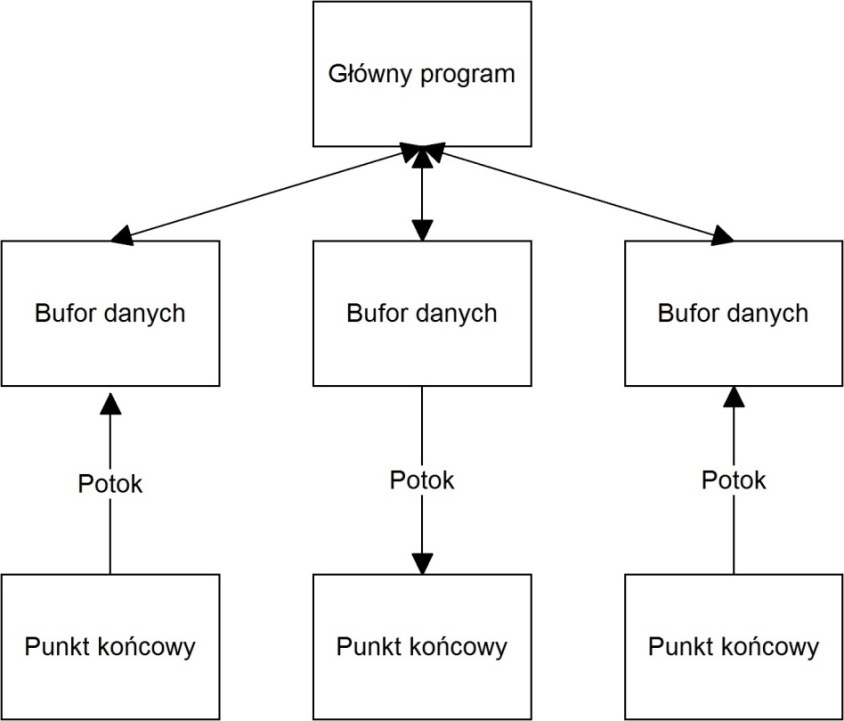
* uniwersalność - posiadając różne typy oraz prędkości przesyłu danych można optymalnie dostosować projekt do wymagań i wykonywanych zadań,
* USB jest wspierane na systemach opartych o jądro GNU/Linux, Windowsie oraz komputerach firmy Apple,
* liczna literatura w postaci opracowań, mnogość przykładów, wsparcie producentów (Cypress).

## 2.1. Wybrane zagadnienia teoretyczne

W niniejszym podrozdziale zostały opisane najważniejsze pojęcia i terminy związane z działaniem magistrali USB na podstawie standardu 2.0.

### 2.2.1. Punkty końcowe oraz potoki

Transfer danych przy użyciu magistrali USB zachodzi pomiędzy głównym oprogramowaniem komputera (ang. host software) a jednostką logiczną urządzenia peryferyjnego określoną mianem punktu końcowego (ang. endpoint) poprzez potoki (ang. pipes). Urządzenie wykorzystujące USB może posiadać maksymalnie 32 aktywne potoki, 16 obsługujących transfer danych do komputera, a druga połowa zarządzająca danymi z urządzenia nadrzędnego [4].



Rys. . Zasada współdziałania punktów końcowych oraz potoków

### 2.2.2. Deskryptory

Urządzenie nadrzędne wykrywa urządzenie peryferyjne na podstawie deskryptorów, które są określane mianem łańcuchów informacji (bajtów danych). Zawierają one informacje na temat oferowanych funkcji przez urządzenie czy liczbę punktów końcowych. Pierwsze 2 bajty zawsze opisują typ deskryptora oraz ilość danych, które zawiera.

Typowe urządzenie peryferyjne posiada:

* deskryptory punktów końcowych,
* deskryptory konfiguracyjne,
* deskryptor urządzenia,
* deskryptory interfejsu.

Deskryptor urządzenia (może być tylko jeden) zawiera nagłówek pakietów (ang. Packet ID; PID określa typ przesyłanego pakietu, czyli jego format i rodzaj kontroli błędów), identyfikator VID (Vendor ID) oraz określa standardowy tryb pracy USB (prędkość). Dla komputera numery PID oraz VID są niezbędne do zainstalowania poprawnych sterowników przez system operacyjny w celu obsługi urządzenia.

Na omówienie zasługuje również deskryptor punktów końcowych. Służy on do przechowywania informacji o typie i kierunku transferu czy maksymalnej wielkości pakietu dla każdego punktu końcowego. Wyjątek stanowi punkt końcowy o numerze 0. Nie posiada on żadnego deskryptora, zawsze jest skonfigurowany jako punkt kontrolny.

### 2.2.3. Typy transferów USB

Standard USB określa cztery typy transferu danych przez potoki [4].

* **przekaz masowy** (ang. bulk transfer) - możliwy jest transfer pakietów zawierających 8, 16, 32 lub 64 bajty danych przy prędkości *full speed* oraz pakietu 512 bajtów danych przy wyższych prędkościach. Ten tryb gwarantuje poprawny transfer danych ze względu na automatyczną retransmisję w przypadku powstania jakichkolwiek błędów,
* **przekaz przerwaniowy** (ang. interrupt transfer) - tryb zbliżony do przekazu masowego, może zawierać pakiety od 1 do 64 bajtów danych przy prędkości *full speed* oraz maksymalnie 1024 bajty przy wyższych prędkościach. Przekaz przerwaniowy polega na tym, że łączność z danym punktem końcowym w urządzeniu peryferyjnym jest nawiązywana okresowo, w określonym wcześniej odstępie czasu,
* **przekaz izochroniczny** (ang. isochronous transfer) - zapewnia stałą szybkość transmisji danych niezależnie od wielkości ruchu generowanego na magistrali. Przekaz izochroniczny stosuje się tam, gdzie wymagany jest stały przepływ pakietów docierających do odbiornika w równych odstępach czasu. Przy zastosowaniu prędkości *full speed* pakiet może zawierać do 1023 bajtów danych oraz maksymalnie 1024 bajty przy prędkości *high speed*,
* **przekazy sterujące** (ang. control transfers) - tryb przewidziany do konfiguracji oraz wysyłania komend sterujących do urządzenia. Jest bardzo ważny z punktu widzenia magistrali, zatem zawiera najbardziej skomplikowane metody sprawdzania błędów transmisji. Urządzenie nadrzędne (ang. host) rezerwuje część każdej ramki USB dla przekazów sterujących.

### 2.2.4. Detekcja i enumeracja urządzenia

Magistrala USB posiada zasadniczą przewagę nad innymi interfejsami, a mianowicie oferuje użytkownikowi możliwość pracy urządzenia z USB na zasadzie *podłącz i używaj* (ang. plug and play). Jeśli urządzenie peryferyjne zostało podłączone po raz pierwszy do sieci wtedy urządzenie nadrzędne analizuje informacje zawarte w jego deskryptorach. Ten proces określa się enumeracją urządzenia [4].

## 2.2. Standard USB 2.0

System USB jest asynchroniczną magistralą komunikacyjną posiadającą głównego hosta oraz do 127 urządzeń peryferyjnych. Host zarządza magistralą, jest on odpowiedzialny za wykrywanie dołączonych urządzeń, inicjalizuje i zarządza transferem danych. Można rozszerzyć liczbę urządzeń peryferyjnych dodając HUB-a w topologii gwiazdy. Standard USB 2.0 wykorzystuje dwa przewody sygnałowe w trybie half-duplex (odbieranie i przesyłanie danych odbywa się na przemian co powoduje niższy transfer) [4].

Wspierane są trzy prędkości przesyłu:

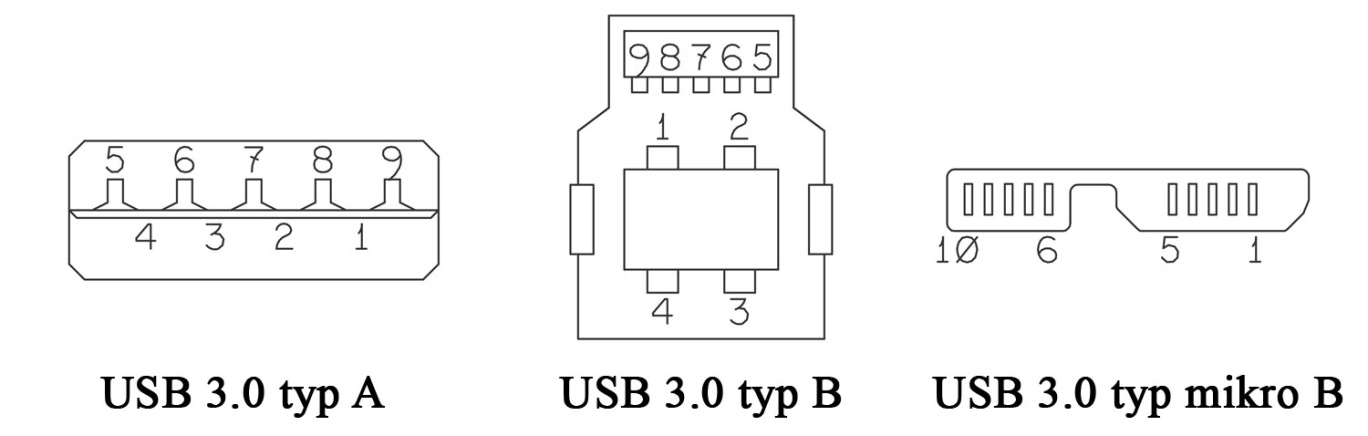
* low speed - transfer z maks. prędkością 1,5 Mbit/s, opracowany w standardzie USB 1.0,
* full speed - transfer z maks. prędkością 12 Mbit/s, opracowany w standardzie USB 1.1,
* high speed - transfer z maks. prędkością 480 Mbit/s, opracowany w 2001 roku (USB 2.0).

## 2.3. Standard USB 3.0

USB 3.0 zawiera dużą liczbę ulepszeń względem poprzedniego standardu, jednakże nie rezygnuje ze wsparcia dla starszej wersji. USB 3.0 wprowadziło nową prędkość transferu *super speed*, która umożliwia przesyłanie danych z prędkością aż do 5 Gbit/s. Jeśli urządzenie główne nie wspiera nowego standardu wtedy urządzenie peryferyjne działa z prędkością *high speed* oferowaną przez USB 2.0. Ponadto USB 3.0 opiera się na wielu identycznych elementach z poprzedniego standardu, np. topologii magistrali, deskryptorach czy prędkościach transferu. Można wyciągnąć wnioski, że USB 3.0 jest w pełni kompatybilną nakładką, dodatkiem do standardu USB 2.0, a nie następcą mającym na celu wyparcie z rynku poprzedni standard. Dzięki znacznemu zwiększeniu przepustowości USB 3.0 znalazło zastosowanie zwłaszcza w urządzeniach magazynujących dane (przenośne dyski twarde) oraz kamerach wideo (obraz w wysokiej rozdzielczości) [3].

W podpunktach wyszczególniono najważniejsze różnice pomiędzy USB 3.0 oraz USB 2.0 [3] [4]:

* dziesięciokrotnie zwiększona przepustowość,
* każdy kierunek transmisji posiada własną parę przewodów, zatem możliwy jest transfer danych w obu kierunkach jednocześnie bez spadku transferu z przepustowością 400 MB/s,
* natężenie prądu wynosi 900 mA, USB 2.0 oferuje maksymalnie 500 mA,
* gniazda oraz konektory posiadają pięć dodatkowych wyprowadzeń; poza standardową parą sygnałów różnicowych (D+ i D-) oraz linii zasilania (+5V i masa) dodatkowo zamieszczono dla USB 3.0: para sygnałów różnicowych do wysyłania danych, para sygnałów różnicowych do odbioru danych oraz dodatkowa linia masy.



Rys. . Złącza wykorzystywane w standardzie USB 3.0 [3]

W zestawie CYUSBKIT-001 dostępne są złącza typu A oraz mikro B (Rys. 2). Poniżej zamieszono ich opis (Tab. 2) [3].

Tab. . Opis złącz wykorzystywanych w USB 3.0

|  |  |  |  |
| --- | --- | --- | --- |
| **Wyprowadzenie** | **Typ A** | **Typ B** | **Typ Mikro B** |
| 1 | +5V | +5V | +5V |
| 2 | D- (USB 2.0) | D- (USB 2.0) | D- (USB 2.0) |
| 3 | D+ (USB 2.0) | D+ (USB 2.0) | D+ (USB 2.0) |
| 4 | masa | masa | identyfikacja USB OTG |
| 5 | RX- (USB 3.0) | TX- (USB 3.0) | masa |
| 6 | RX+ (USB 3.0) | TX+ (USB 3.0) | TX- (USB 3.0) |
| 7 | masa | masa | TX+ (USB 3.0) |
| 8 | TX- (USB 3.0) | RX- (USB 3.0) | masa |
| 9 | TX+ (USB 3.0) | RX+ (USB 3.0) | RX- (USB 3.0) |
| 10 | - | - | RX+ (USB 3.0) |

# 3. Interfejs synchronicznej kolejki FIFO oraz programowalna maszyna stanów GPIF II

Jednym z najpopularniejszych interfejsów, który można zaimplementować w urządzeniu EZ-USB FX3 firmy Cypress jest synchroniczna kolejka FIFO. Ten rodzaj transmisji posiada bardzo bogatą dokumentację dołączoną do zestawu CYUSBKIT-001 na płycie CD oraz dostępną na oficjalnej stronie producenta [7]. Bezpośredni dostęp do rejestrów FX3 nie jest możliwy w tym zastosowaniu, zatem urządzenie zewnętrzne (w tym przypadku Spartan 3E) zapisuje lub odczytuje dane z buforów FIFO (ang. First In, First Out, pierwszy na wejściu, pierwszy na wyjściu) [9]. Poniżej przedstawiono schemat działania docelowej aplikacji (Rys. 3):



Rys. . Schemat działania aplikacji

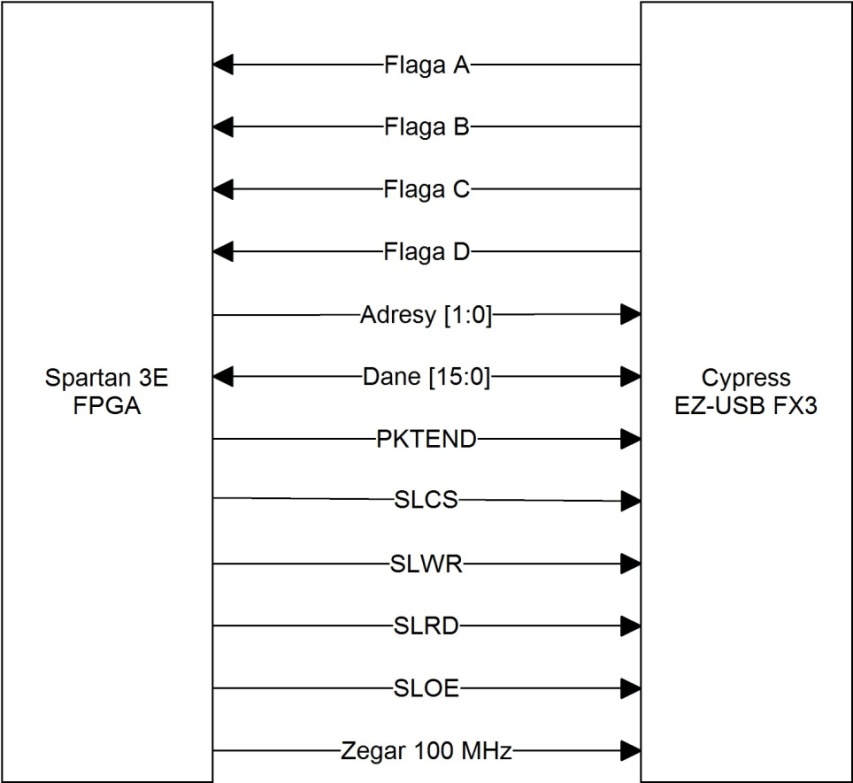
## 3.1. Interfejs synchronicznej kolejki FIFO

Interfejs synchronicznej kolejki FIFO znajduje zastosowanie, gdy zewnętrze urządzenie (dowolny procesor lub układ programowalny) standardowo nie posiada magistrali USB, a zachodzi potrzeba wymiany danych z komputerem. W tym przypadku można zastosować komunikację RS-232, jednak użytkownika ogranicza niska przepustowość magistrali. Urządzenie EZ-USB FX3 rozwiązuje przedstawiony problem dodając do aplikacji możliwość szybkiego transferu danych z wykorzystaniem magistrali USB.

Na Rys. 4 przedstawiono sygnały zastosowane w interfejsie synchronicznej kolejki FIFO. W Tab. 3 zamieszczono ich szczegółowy opis.

Tab. . Sygnały synchronicznej kolejki FIFO

|  |  |
| --- | --- |
| **Nazwa sygnału** | **Opis** |
| Flaga A, B, C, D | Wskazują możliwość dostępu do gniazd FX3 (puste/pełne). Flagi A i B są używane podczas transmisji wysyłania danych, a C i D przy odbiorze. |
| Adresy [1:0] | 2-bitowa szyna adresowa, wskazuje na aktualny wątek |
| Dane [15:0] | 16-bitowa szyna danych (może być 8, 16 lub 32-bitowa) |
| Pktend | Sygnał używany przy transmisji krótkiego pakietu (ang. short packiet) lub o zerowej długości (ang. zero-length packet) |
| SLCS | Wybór urządzenia, sygnał zawsze aktywny podczas dowolnej transmisji |
| SLWR | Sygnał zapisu danych do FX3 |
| SLRD | Sygnał odczytu danych z FX3 |
| SLOE | Sygnał zezwolenia wysyłania danych do FX3 |
| Zegar 100 MHz | Sygnał zegarowy wytworzony w urządzeniu nadrzędnym (FPGA), doprowadzony do urządzenia podrzędnego (FX3) |



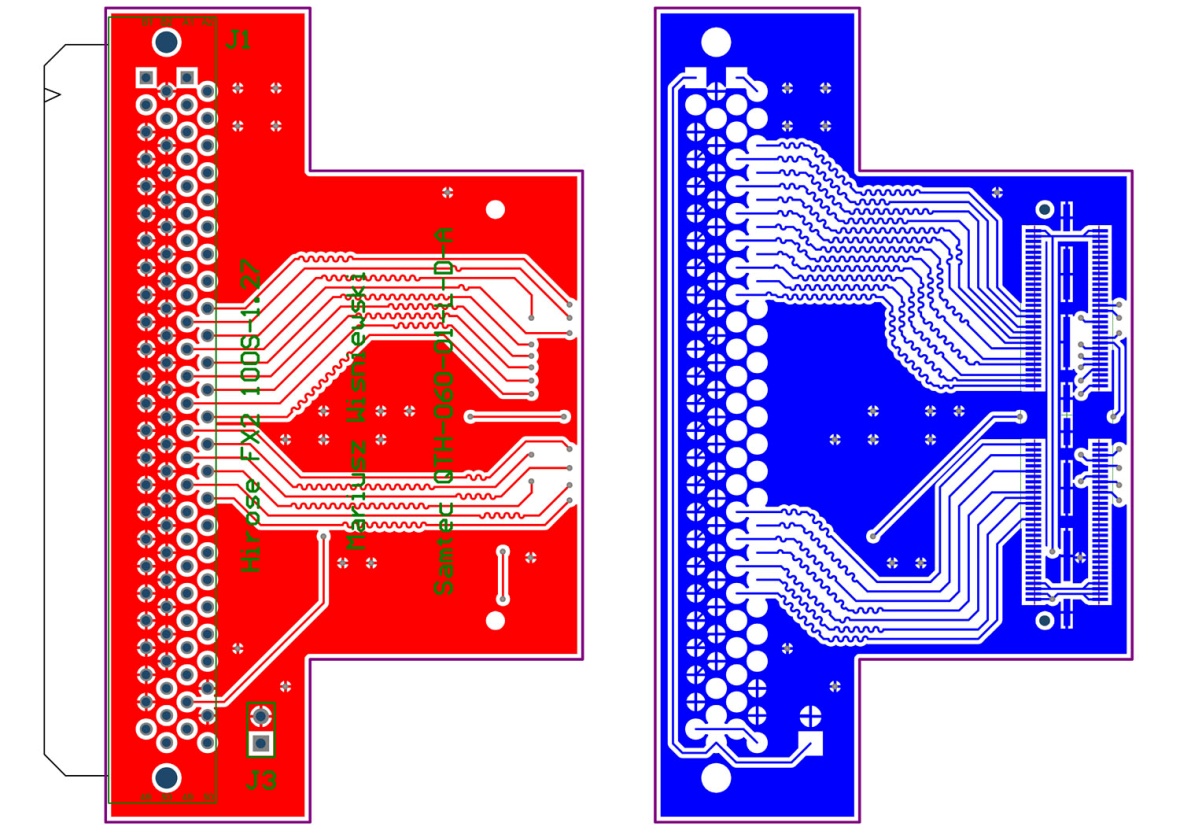
Rys. . Schemat interfejsu synchronicznej kolejki FIFO

Interfejs dzięki 2-bitowej linii adresowej jest w stanie uzyskać dostęp do maksymalnie czterech wątków. Istnieje możliwość rozszerzenia tej liczby, jednak wiąże się to z potrzebą użycia 5-bitowej linii adresowej. Pociąga to za sobą konsekwencje, m.in.: dodatkowe opóźnienia podczas monitorowania stanu flagi oraz po zmianie adresu [9]. W omawianej pracy nie rozpatrywano przypadku użycia 5-bitowej linii adresowej.

## 3.2. Projekt obwodu drukowanego łączącego dwa zestawy uruchomieniowe

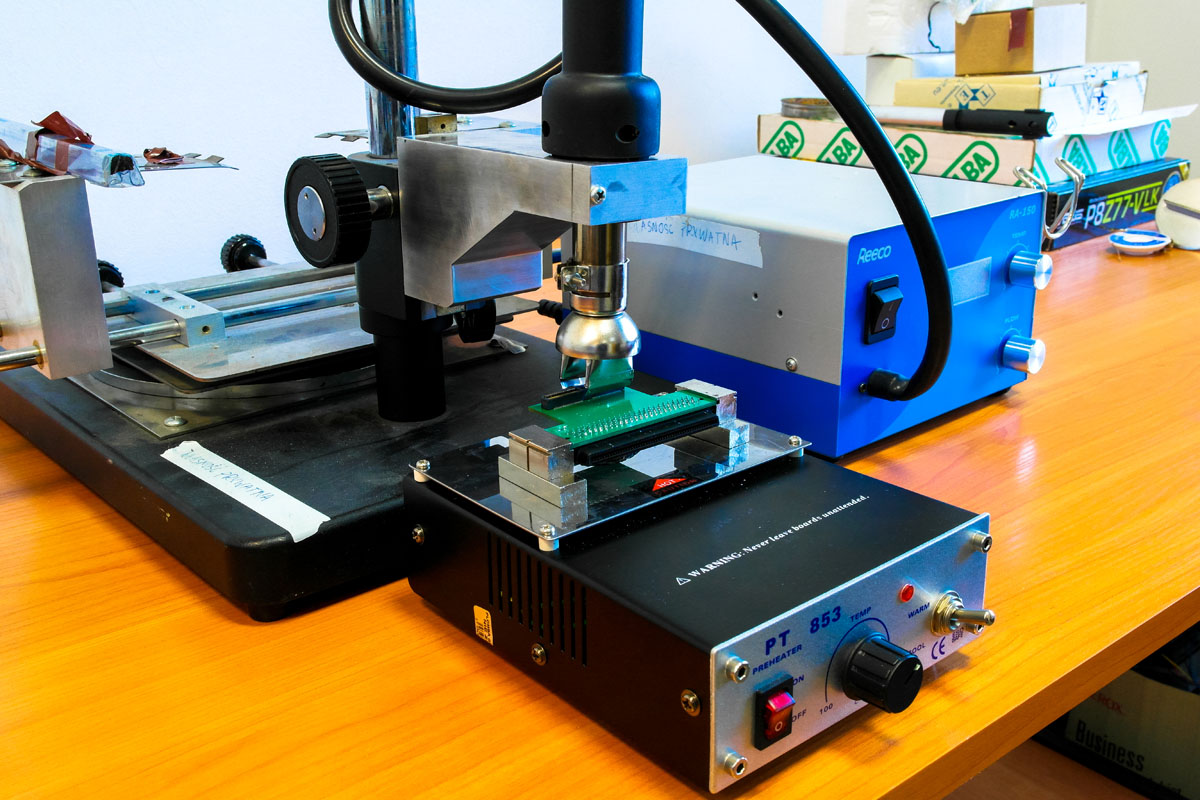
Firma Cypress nie posiada w swojej ofercie obwodu drukowanego, który umożliwiłby na sprzęgnięcie ze sobą używanych zestawów w pracy. W sklepie internetowym producenta FX3 można odnaleźć jedynie produkt o nazwie CYUSB3ACC-002 (zaprzestano jego produkcji, niedostępny w sprzedaży). Ponadto przeznaczony jest on jedynie dla zestawów uruchomieniowych wyposażonych w złącze FMC (Xilinx LPC).

Schemat elektryczny oryginalnej płytki [17] oraz dokumentacja elektryczna zestawu Spartan 3E [16] posłużyły jako baza do zaprojektowania własnej przejściówki, opartej na złączach Hirose FX2 oraz Samtec QTH-060-01-L-D-A (Zał. 1). Projekt PCB (Rys. 5) został wykonany w programie Altium Designer, ze względu na możliwości, które oferuje to oprogramowanie. Wszystkie ścieżki (oprócz linii zegarowej) posiadają identyczną długość. Umożliwia to zlikwidowanie ewentualnych opóźnień na magistrali danych i eliminację efektu propagacji sygnałów. Zaprojektowana płytka składa się z dwóch warstw.



Rys. . Projekt obwodu drukowanego. Z lewej strony warstwa górna, z prawej strony warstwa dolna

Po wygenerowaniu w Altium Designerze wszystkich warstw i otworów projekt został wysłany do firmy Techno-Service, która specjalizuje się w wykonywaniu obwodów drukowanych. Po otrzymaniu gotowej płytki rozpoczęto przygotowania do montażu złącza QTH-060-01-L-D-A metodą na gorące powietrze zgodnie z zaleceniami producenta [18].

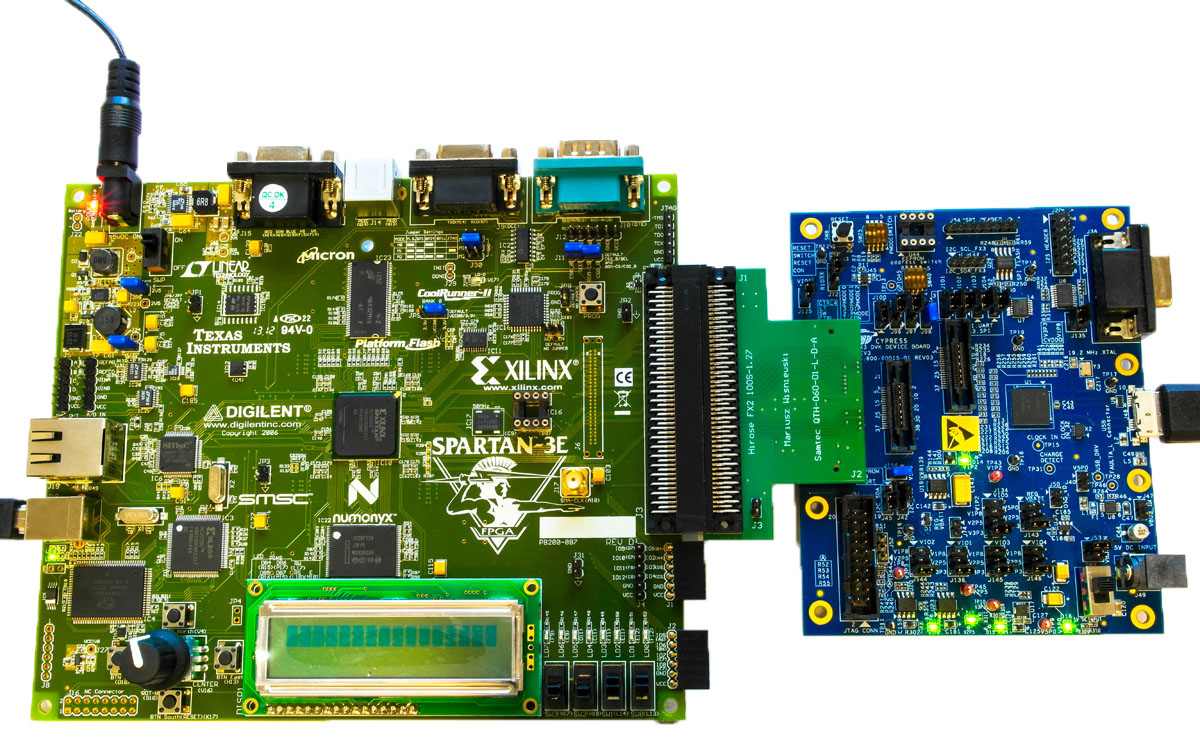


Rys. . Proces montażu elementów SMD gorącym powietrzem, widoczny statyw, podgrzewacz oraz lutownica hot-air z odpowiednimi dyszami

Do lutowania metodą na gorące powietrze posłużyły następujące narzędzia (Rys. 6):

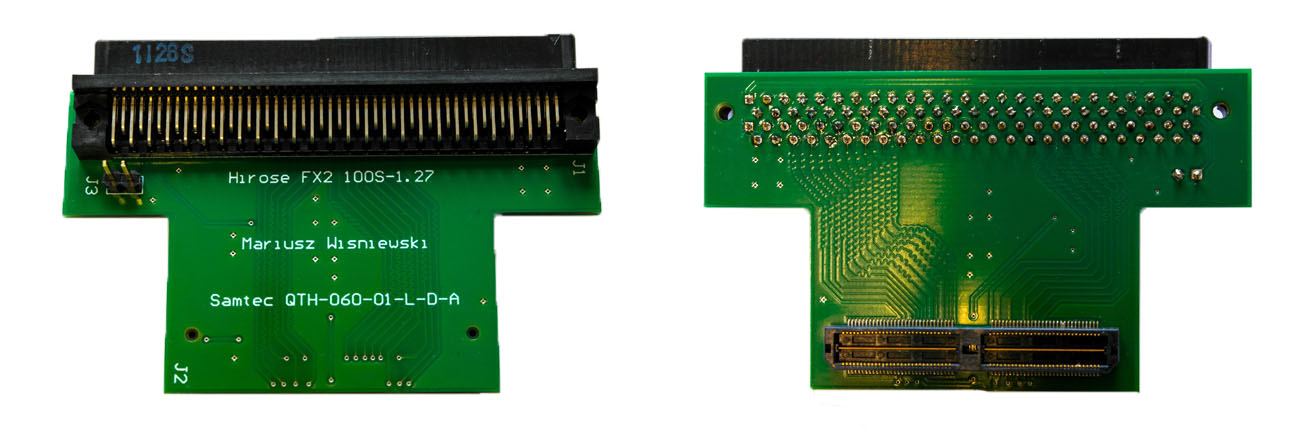
* podgrzewacz PT 853,
* stacja hot-air Reeco RA-150 wraz z odpowiednimi dyszami,
* statyw Sunko 301.

Poniżej przedstawiono fotografię efektu końcowego, czyli gotową płytkę sprzęgającą dwa zestawy uruchomieniowe (Rys. 7).



Rys. . Płytka PCB łącząca Spartana 3E oraz EZ-USB FX3

Zdjęcia wykonanego obwodu drukowanego zamieszczono na Rys. 8:



Rys. . Po lewej złącze Hirose FX2, po prawej Samtec QTH-060-01-L-D-A

## 3.3. Programowalna maszyna stanów GPIF II

GPIF II jest programowalną maszyną stanów, która wprowadza możliwość implementacji dowolnego standardu komunikacji (własnego lub wcześniej opracowanego przez korporacje). Może funkcjonować jako podmiot nadrzędny (ang. master) lub podrzędny (ang. slave). W pracy GPIF II podlega urządzeniu nadrzędnemu, czyli FPGA. Funkcje posiadane przez GPIF II:

* funkcjonuje jako urządzenie nadrzędne lub podrzędne,
* oferuje 256 programowalnych stanów,
* wspiera 8, 16 i 32-bitową równoległą szynę danych,
* maksymalna częstotliwość pracy wynosi 100 MHz,
* możliwość konfiguracji 14 niezależnych sygnałów przy 32-bitowej szynie danych,
* możliwość konfiguracji 16 niezależnych sygnałów przy 8 lub 16-bitowej szynie danych.

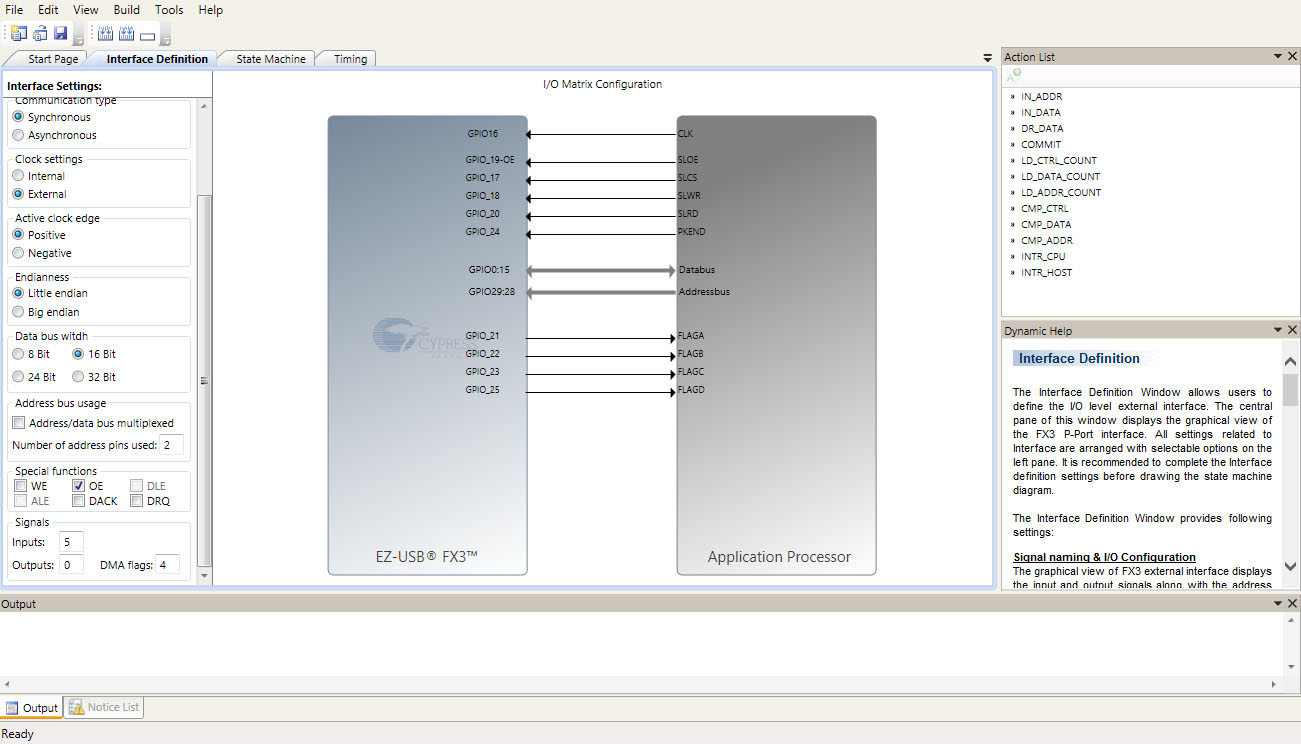
Zachowanie maszyny stanów zostało zdefiniowane w deskryptorze, który zawiera zestaw programowalnych rejestrów konfiguracyjnych. W kontrolerze EZ-USB FX3 przydzielono mu 8 kB pamięci [9].

## 3.4. Narzędzie GPIF II Designer

W niniejszym rozdziale przedstawiono konfigurację interfejsu synchronicznej kolejki FIFO w programie GPIF II Designer, począwszy od rozpoczęcia projektu aż do konfiguracji częściowych flag (B i D) [10].

### 3.4.1. Zdefiniowanie i konfiguracja sygnałów interfejsu

Rozpoczęto od importu projektu *sync\_slave\_fifo\_2bit* przygotowanego przez firmę Cypress. Po załadowaniu plików do programu nie było możliwości zmiany parametrów interfejsu. Należało zapisać projekt jako edytowalny (*File - Save Project As Editable*). Po otworzeniu właściwego projektu dostępne stały się pełne możliwości edycyjne programu GPIF II Designer.



Rys. . Okno programu GPIF II Designer

W lewej części okna skonfigurowano odpowiednie opcje, które dostosowały projekt do założeń interfejsu synchronicznej kolejki FIFO (Rys. 9). Przeprowadzono następujące zmiany:

* typ interfejsu: podrzędny (ang. slave),
* komunikacja: synchroniczna,
* sygnał zegarowy: zewnętrzny,
* reakcja na zbocze: stan wysoki,
* kolejność bajtów: cienkokońcowość (little endian),
* liczba bitów szyny danych: 16,
* liczba bitów szyny adresowej: 2,
* sygnałów wejściowych: 5,
* flag DMA: 4.

Następnie zmieniono domyślne nazwy sygnałów na docelowe (np. SLCS, FLAGA) dla zwiększenia czytelności projektu. Dla każdego sygnału przyjęto osobną konfigurację oraz przypisano mu odpowiednie wyprowadzenie na procesorze FX3. W Tab. 4 zawarto zmiany przeprowadzone dla poszczególnych sygnałów:

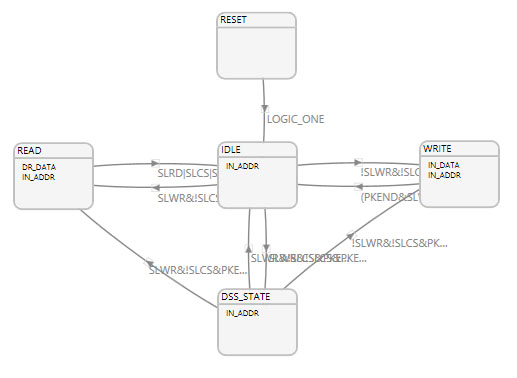
Tab. . Konfiguracja sygnałów interfejsu

|  |  |  |
| --- | --- | --- |
| **Nazwa sygnału** | **Logiczne "1"** | **Wyprowadzenie FX3** |
| CLK | Stan niski | 16 |
| SLOE | 19 |
| SLCS | 17 |
| SLWR | 18 |
| SLRD | 20 |
| PKTEND | 24 |
| Databus (n-bitowa szyna danych) | Nie dotyczy | 0:15 |
| Addressbus (szyna adresowa) | Nie dotyczy | 29:28 |
| FLAGA | Stan niski | 21 |
| FLAGB | 22 |
| FLAGC | 23 |
| FLAGD | 25 |

### 

### 3.4.2. Konfiguracja maszyny stanów

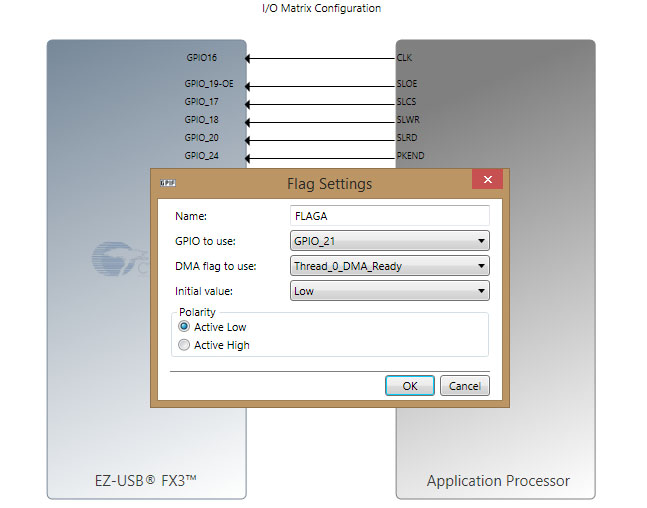
Oryginalna maszyna stanów opracowana przez firmę Cypress obejmowała 7 stanów. Po przeprowadzeniu modyfikacji zmniejszono tę liczbę do 5 stanów (Rys. 10).



Rys. . Maszyna stanów programowalnej maszyny stanów GPIF II

### 3.4.3. Konfiguracja flag pełnych oraz częściowych

Dla interfejsu synchronicznej kolejki FIFO skonfigurowano 4 flagi. Flagi A i C ustawiono jako flagi pełne, natomiast B i D jako flagi częściowe.



Rys. . Przyporządkowanie flagi do wątku

Każdej z poszczególnych flag przyporządkowano odpowiadający jej wątek. Dokonano tego wybierając opcję ustawienia flagi *(DMA Flag Settings)* po wcześniejszym skierowaniu kursora na jej nazwę oraz wybraniu prawego przycisku myszy (Rys. 11). Tab. 5 przedstawia konfigurację wszystkich czterech flag.

Tab. . Konfiguracja flag w GPIF II Designerze

|  |  |  |  |
| --- | --- | --- | --- |
| **Wyprowadzenie FX3** | **Nazwa sygnału** | **Logiczne "1"** | **Wątek** |
| 21 | Flaga A | Stan niski | Thread\_0\_DMA\_Ready |
| 22 | Flaga B | Thread\_0\_DMA\_Watermark |
| 23 | Flaga C | Thread\_3\_DMA\_Ready |
| 25 | Flaga D | Thread\_3\_DMA\_Watermark |

### 3.4.4. Kompilacja projektu

Po dokonaniu wszystkich modyfikacji zapisano projekt oraz dokonano jego skompilowania. Powyższe działanie miało na celu wygenerowanie w folderze projektu pliku *cyfxgpif2config.h* zawierającego konfigurację programowalnej maszyny stanów.

# 4. Cypress EZ-USB FX3

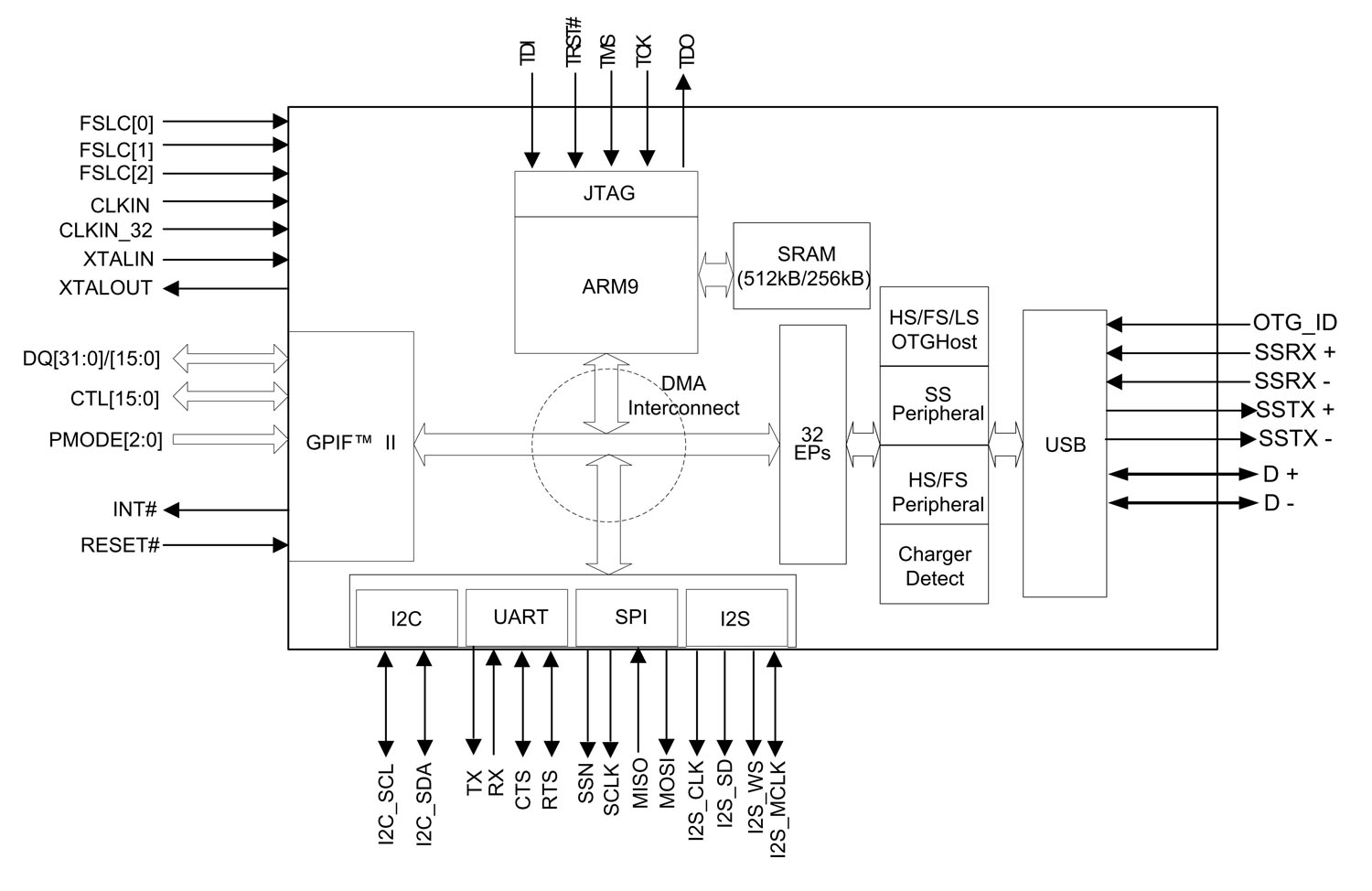
Zastosowanie zestawu ewaluacyjnego EZ-USB FX3 (w dalszej części pracy określany mianem FX3) firmy Cypress umożliwia dodanie funkcjonalności magistrali USB 3.0 do dowolnego urządzenia zewnętrznego (Rys. 12). Procesor FX3 został oparty na architekturze ARM9. Oferuje on łączność z zewnętrznymi urządzeniami poprzez magistrale zarówno szeregowe jak i równoległe.



Rys. . Zestaw Cypress EZ-USB FX3 (CYUSB3KIT-001) [7]

Główną funkcją FX3 jest transfer danych o wysokiej przepustowości pomiędzy hostem USB (komputerem), a urządzeniem zewnętrznym, np. FPGA, kamerą czy procesorem sygnałowym. Moc procesora opartego na rdzeniu ARM9 pozwala również na manipulację danymi bez większych obciążeń. Jednakże nie wykorzystano wszystkich wymienionych możliwości, gdyż celem nadrzędnym było opracowanie interfejsu bezpośredniej transmisji danych bez jakichkolwiek ingerencji w pakiety [4].

Poza standardowymi magistralami takimi jak I2C, SPI, UART czy I2S FX3 oferuje programowalną maszynę stanów GPIF II (II w nazwie oznacza drugą generację przygotowaną przez firmę Cypress). Jest ona w pełni elastyczna, umożliwia implementację interfejsów zarówno szeregowych, jak i równoległych. Złącze QSH-060-L-D-A firmy Samtec pozwala na sprzęgnięcie z interfejsem GPIF II dowolnego zewnętrznego układu, na przykład: procesory 8, 16 lub 32-bitowe, układy programowalne FPGA czy procesory sygnałowe DSP. Schemat budowy FX3 (Rys. 13) został zaczerpnięty z dokumentacji producenta [6].



Rys. . Schemat blokowy FX3

Opis opcji oferowanych przez kontroler USB:

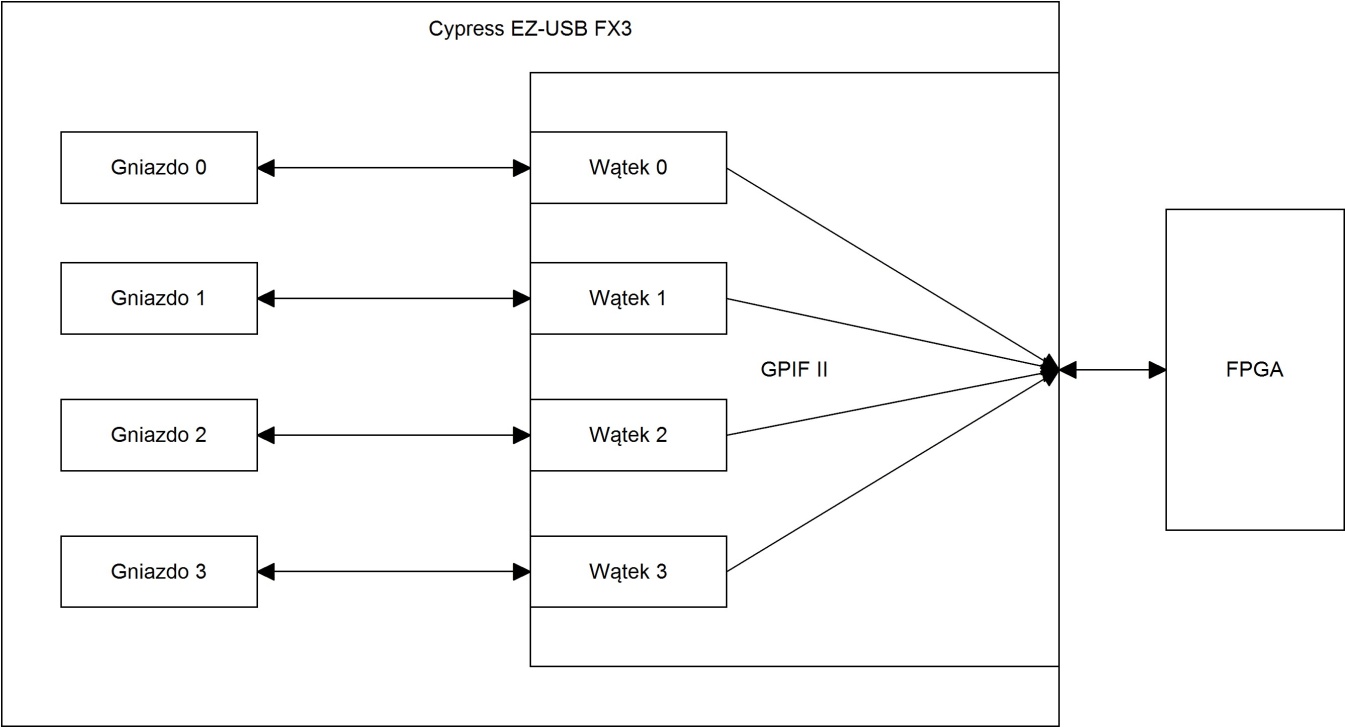
* interfejs USB - kompatybilność z USB 2.0 (wszystkie prędkości, które oferuje standard) oraz z USB 3.0, do 16 wejściowych oraz 16 wyjściowych punktów końcowych (ang. endpoints),
* GPIF II,
* 32-bitowy procesor ARM926EJ-S oparty na rdzeniu ARM9, 512 kB pamięci SRAM
* interfejs JTAG - 5-sygnałowy interfejs umożliwiający debugowanie programu,
* UART - interfejs wspiera komunikacje w trybie full-duplex (dane są przesyłane w obu kierunkach jednocześnie bez spadku transferu), sygnały TX, RX, CTS oraz RTS, prędkość od 300 bit/s do 4608 Kb/s,
* I2C - FX3 działa jako urządzenie nadrzędne (master), zatem pozwala na dołączenie urządzeń podrzędnych (ang. slave). FX3 może pobierać program podczas startu z pamięci EEPROM, która komunikuje się poprzez magistralę I2C. Wspierane są 3 częstotliwości zegarowe: 100 kHz, 400 kHz oraz 1 MHz,
* I2S - magistrala umożliwiająca przesyłanie danych audio, służy do podłączania zewnętrznych urządzeń nadawczych lub odbiorczych,
* SPI - magistrala operująca na maksymalnej częstotliwości taktowania 33 MHz,
* zasilanie - każdy blok peryferyjny może być zasilany z osobnego źródła o różnym napięciu,
* sygnał zegarowy - wewnętrzny zegar taktuje z częstotliwością 19.2 MHz, jest możliwość dołączania zewnętrznych zegarów do 52 MHz.

## 4.1. Teoria FX3 - wątki, gniazda, DMA oraz deskryptory

Poniższy podrozdział opisuje teoretyczną stronę realizacji synchronicznej kolejki FIFO oraz mechanizmy wykorzystywane przez EZ-USB FX3 [9].

### 4.1.1. Gniazda

Gniazdo jest punktem połączeniowym pomiędzy blokiem peryferyjnym a pamięcią RAM FX3 (Rys. 14). Każdy blok (USB, GPIF, UART czy SPI) posiada tyle związanych z nim gniazd, ile zdefiniowano kierunków transferów danych (np. jedno gniazdo jest odpowiedzialne za transfer danych z USB do GPIF, a drugie z GPIF do USB).



Rys. . Schemat działania gniazd oraz wątków. W danym momencie z FPGA może współpracować tylko jeden wątek, pozostała część jest nieużywana [6]

### 4.1.2. Deskryptory DMA

Pojedynczy deskryptor DMA (ang. Direct Memory Access, czyli bezpośredni dostęp do pamięci) jest zestawem rejestrów umieszczonych w pamięci RAM FX3. Zawiera on informacje o adresach, wielkości buforów DMA oraz wskaźniki do następnych deskryptorów. Komplet wskaźników tworzy łańcuch deskryptorów DMA.

### 4.1.3. Bufor DMA

Bufory DMA są zaalokowane w pamięci RAM, ich adresy przechowują deskryptory DMA. FX3 używa buforów do pośredniego przechowywania danych przed ich transferem w obu kierunkach.

### 4.1.4. Wątki GPIF II

Wątek GPIF II to dedykowana ścieżka danych w bloku maszyny stanów, który łączy zewnętrzne peryferia z GPIF-em poprzez gniazda (Rys. 14). Gniazda mogą się komunikować dzięki zdarzeniom (ang. events) oraz przerwaniom procesora FX3 (ang. interrupts), ich konfiguracja odbywa się w oprogramowaniu.

**Przykład**: niech następuje transfer danych z bloku GPIF II do bloku USB. Gniazdo GPIF może poinformować gniazdo USB, że zostało zapełnione danymi w buforze DMA oraz gniazdo USB może poinformować o dostępnym wolnym miejscu w dedykowanym buforze DMA. Taka implementacja jest nazwana automatycznym kanałem DMA, procesor nie bierze udziału w jakichkolwiek zmianach danych podczas przesyłu.

Jednakże gniazdo GPIF jest w stanie zasygnalizować wywołanie przerwania przez procesor, gdy bufor DMA zostanie zapełniony. Wtedy procesor odpowiada za przekazanie tej informacji gniazdu USB. W odpowiedzi gniazdo USB może przekazać do procesora sygnał o wywołaniu przerwania informującego o tym, że bufor DMA jest pusty. To działanie pociąga za sobą ponowne poinformowanie gniazda GPIF przez CPU o dostępności gniazda USB. Opisana implementacja nosi nazwę manualnego kanału DMA. Używany jest, gdy procesor ma za zadanie modyfikować dane.

Gniazdo, które zapisuje dane do bufora DMA nazwano producentem. Z kolei gniazdo, które z niego odczytuje dane zostało określone konsumentem. Gniazdo używa takich wartości jak: adres i wielkość buforu DMA oraz łańcucha deskryptorów DMA w celu zarządzania danymi. Zmiana deskryptora z jednego na kolejny przez gniazdo trwa kilka mikrosekund, w tym czasie nie jest możliwy żaden transfer danych.

EZ-USB FX3 umożliwia wykorzystanie czterech wątków dla transferu danych poprzez blok GPIF II. Tylko jedno gniazdo może być wykorzystywane przez pojedynczy wątek. Standardowo nazwa wątków jest ściśle związana z nazwami gniazd, np. gniazdo 0 - wątek 0, gniazdo 4 - wątek 4.

Linia adresowa (A1:A0) interfejsu synchronicznej kolejki FIFO wskazuje na aktualnie używane wątek. Oprogramowanie zostało tak przygotowane przez firmę Cypress, że zmiana wątku automatycznie pociąga za sobą zmianę gniazda. Zatem jeśli zmieniono adres z 0 na 3 wtedy FX3 przypisuje gniazdo 3 do wątku o identycznym numerze.

### 4.1.5. Konfiguracja kanału DMA

Konfigurację kanału DMA przeprowadza się w oprogramowaniu przypisując do niego gniazda producenckie (źródła danych) oraz konsumenckie (pobierające dane).

**Przykład**: jeśli następuje transfer danych z bloku GPIF do USB to wtedy blok GPIF jest nazywany producentem, a USB konsumentem.

Zatem jeśli zachodzi transfer w obu kierunkach należy skonfigurować dwa kanały DMA, gdzie GPIF jest blokiem zarówno producenckim jak i konsumenckim (analogicznie USB).

Należy dodać, że do pojedynczego kanału DMA można przypisać kilka buforów. Zasada wskazywania pełnego lub pustego buforu przez flagi została opisana w przykładzie poniżej.

**Przykład**: niech dwa bufory o pojemności 1024 bajtów zostaną przypisane do konkretnego kanału DMA, wtedy pełna flaga będzie wskazywać zapełnienie bufora, gdy 1024 bajtów danych zostanie zapisanych do pierwszego bufora. Flaga dalej będzie wskazywać brak wolnego miejsca do zapisu dopóki kanał DMA nie przełączy się na drugi bufor. Czas zmiany bufora nie jest określony w dokumentacji (nie jest stały), jednakże trwa nie więcej niż 10 mikrosekund. Urządzenie nadrzędne (ang. master, w pracy FPGA) ma za zadanie nieprzerwanie monitorować stan flagi, by wiedzieć kiedy zakończono przełączenie się do pustego buforu.

## 4.2. Konfiguracja flag pełnych oraz częściowych

Flagi mogą być skonfigurowane jako puste, pełne, częściowo puste lub częściowo pełne. Nie są one sterowane przez programowalną maszynę stanów GPIF II, lecz przez kanały DMA procesora FX3. Flaga jest przypisana do konkretnego lub do aktualnie adresowanego wątku oraz wskazuje status gniazda związanego z wątkiem.

Flagi wskazują stan pusty albo pełny zależnie od kierunku, w którym gniazdo przesyła dane. Zatem flaga wskazuje stan pusty / częściowo pusty jeśli dane są odczytywane z gniazda lub status pełny / częściowo pełny gdy dane są zapisywane do gniazda.

Wyróżnia się dwa typy flag:

* flaga dedykowana do wątku (pusta / pełna lub częściowo pusta / pełna)
* flaga wskazująca na aktualny wątek (pusta / pełna lub częściowo pusta / pełna)

Użycie konkretnego typu wiąże się z uzyskaniem różnych opóźnień, które należy uwzględnić w oprogramowaniu.

### 4.2.1. Flagi dedykowane, związane z konkretnym wątkiem

Flaga dedykowana wskazuje status tylko jednego, konkretnego wątku. Innymi słowy, zawsze wskazuje stan przypisanego wątku niezależnie od tego, który wątek jest aktualnie wykonywany i adresowany. Zatem urządzenie nadrzędne jest zobowiązane jedynie do śledzenia poprawnej flagi, gdy zmieniono wątek przy użyciu linii adresowej. Opóźnienia związane z flagami podczas odczytu i zapisu danych zostały umieszczone w Tab. 6.

**Przykład**: jeśli flaga A została przypisana do wątku o numerze 0, a flagę B związano z wątkiem 1, urządzenie nadrzędne podczas zmiany stanu linii adresowej na 0 monitoruje jedynie stan flagi A, a po zmianie adresu na 1 przestaje monitorować flagę A i zaczyna obserwować sygnał flagi B.

### 4.2.2. Flagi wskazujące na aktualny wątek

W omawianym trybie flagi wskazują stan aktualnie adresowanego wątku. Programowalna maszyna stanów GPIF II próbkuje dane z linii adresowej oraz na tej podstawie aktualizuje flagę, aby wskazywała na konkretny wątek. Taka konfiguracja nie angażuje dodatkowych linii w interfejsie synchronicznej kolejki FIFO, więc wystarczy jedynie 1 flaga do poprawnego pokazywania stanów 4 wątków. Generowane jest opóźnienie w postaci 2 taktów zegara, ponieważ GPIF w pierwszej kolejności musi sprawdzić stan na linii adresowej, a dopiero potem zaktualizować flagę.

### 4.2.3. Flagi częściowe

Flaga częściowa wskazuje status częściowo pustego / pełnego gniazda. Dodatkowo należy obliczyć wartość liczby (ang. watermark) niezbędnej do skonfigurowania gniazda używając jednej z funkcji API Cypressa.

Tab. 6 przedstawia generowane opóźnienia podczas używania danego trybu konfiguracji flagi. Ponadto zostały zamieszczone przykłady użycia API, które jest niezbędne w celu realizacji flag częściowych.

Tab. . Opóźnienia związane z różnymi typami flag [9]

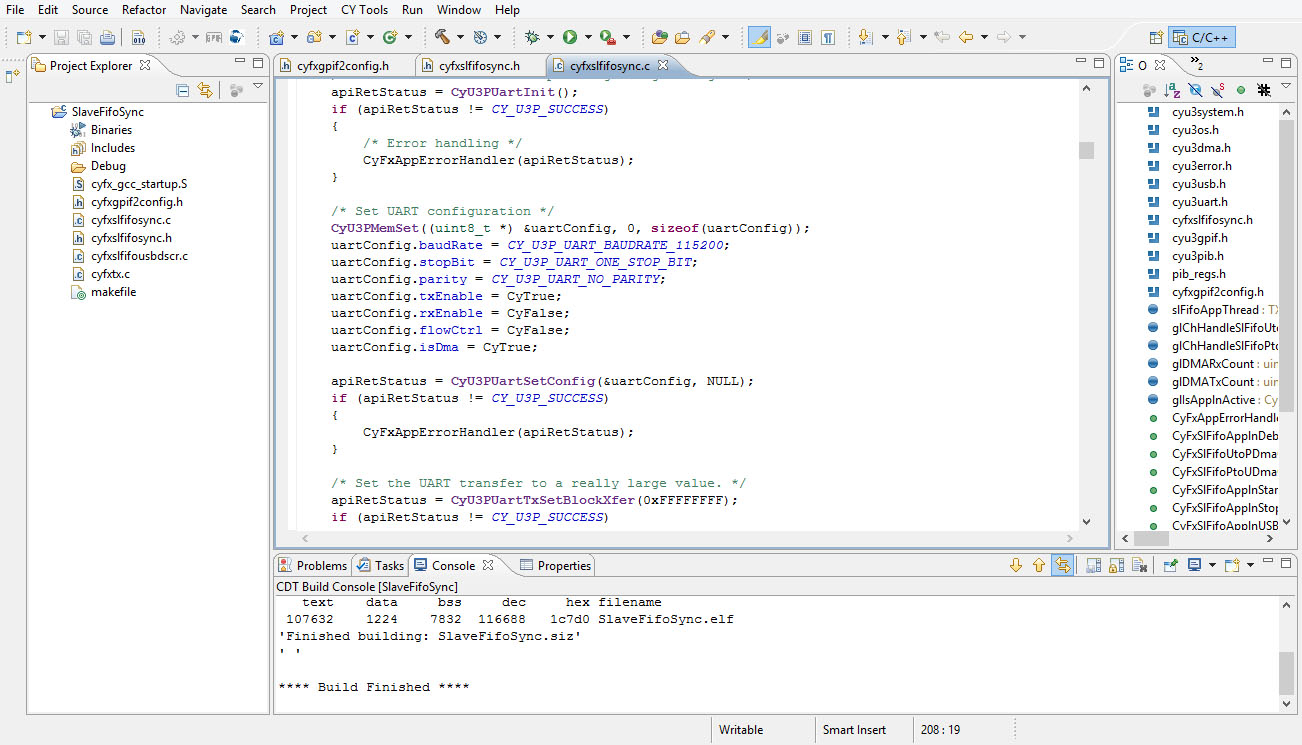
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Typ flagi** | **Opcja w programie GPIF II Designer** | **Opóźnienie związane z adresowaniem** | **Opóźnienie flag na koniec transferu** | | **Wykorzystanie API** |
| **Zapis danych** | **Odczyt danych** |
| Pełna flaga dedykowana do wątku | Thread n DMA Ready | 0 cykli zegara | 3 cykle zegara | 2 cykle zegara | Nie dotyczy |
| Pełna flaga wskazująca na aktualny wątek | Current thread DMA Ready | 2 cykle zegara | 3 cykle zegara | 2 cykle zegara | Nie dotyczy |
| Częściowa flaga dedykowana do wątku | Thread n DMA Watermark | 0 cykli zegara | Zależne od atrybutu "watermark" | Zależne od atrybutu "watermark" | Niezbędne użycie funkcji CyU3SocketConfigure() |
| Częściowa flaga wskazująca na aktualny wątek | Current thread DMA Watermark | 2 cykle zegara | Zależne od atrybutu "watermark" | Zależne od atrybutu "watermark" | Niezbędne użycie funkcji CyU3SocketConfigure() |

## 4.3. Interfejs synchronicznej kolejki FIFO - realizacja oprogramowania dla FX3

Podobnie jak w przypadku narzędzia GPIF II Designer firma Cypress przygotowała przykładowy projekt, który posłużył jako baza do zrealizowania interfejsu synchronicznej kolejki FIFO [9].

### 4.3.1. Zintegrowane środowisko programistyczne Cypress EZ-USB Suite

Cypress udostępnia pełen pakiet oprogramowania na płycie CD dołączonej do zestawu, jednakże pobrano z oficjalnej strony producenta najnowszą wersję oprogramowania. Po instalacji uruchomiono zintegrowane środowisko programistyczne (ang. IDE) o nazwie EZ-USB Suite oparte na popularnej platformie Eclipse. Pierwszym krokiem było zaimportowanie przykładowego projektu o nazwie *SlaveFifoSync*.



Rys. . Okno środowiska EZ-USB Suite

Szata graficzna środowiska nie odbiega standardem od większości IDE (Rys. 15). Po lewej stronie znajduje się drzewo plików projektu, w środku okna umieszczono karty z otwartymi plikami źródłowymi, natomiast po prawej stronie widnieje lista zmiennych i funkcji użytych w programie. Na dole są umieszczone karty zawierające m.in. konsolę czy informacje na temat problemów powstałych podczas kompilacji projektu.

Postanowiono utworzyć dwa projekty, jeden odpowiedzialny za strumieniowy transfer danych, a drugi obsługujący transfer danych w obu kierunkach. Każdy z projektów posiada osobną konfigurację oraz różne stałe związane z transferem danych.

### 4.3.2. Tryb jednokierunkowego wysyłania lub odbierania danych

Dołączono plik *cyfxgpif2config.h* do projektu, który został wcześniej wygenerowany w programie GPIF II Designer. Zwiera on konfigurację programowalnej maszyny stanów, definicje użytych sygnałów w interfejsie oraz ich polaryzację (logiczne "1" to stan niski).

Plik nagłówkowy *cyfxslfifosync.h* jest zbiorem stałych (Tab. 7), które posłużyły do konfiguracji ustawień magistrali USB wykorzystując funkcje *CyU3PDmaChannelCreate, CyU3PThreadCreate, CyU3PDmaChannelSetXfer* oraz struktury *CyU3PDmaChannelConfig\_t* [5].

Tab. . Zbiór stałych w pliku nagłówkowym

|  |  |  |
| --- | --- | --- |
| **Zmienna** | **Wartość stała** | **Opis** |
| CY\_FX\_SLFIFO\_GPIF\_16\_32BIT\_CONF\_SELECT | 0 lub 1 | Wybór szyny danych: 16-bitowa lub 32-bitowa |
| AUTO\_MANUAL\_CONF\_SELECT | 0 lub 1 | Wybór kanału DMA: automatyczny lub manualny |
| DMA\_BUF\_SIZE | 16 | Wielkość buforu DMA |
| CY\_FX\_SLFIFO\_DMA\_BUF\_COUNT\_P\_2\_U | 8 | Liczba buforów zaalokowanych w kanale. Dla kanałów automatycznych wartość nie może wynosić 0. |
| CY\_FX\_SLFIFO\_DMA\_BUF\_COUNT\_U\_2\_P | 4 |
| CY\_FX\_SLFIFO\_DMA\_TX\_SIZE | 0 | Drugi argument dla funkcji *CyU3PDmaChannelSetXfer*. Pożądana wielkość transferu, kanał powróci do stanu bezczynności po przesłaniu określonej liczbie danych. Wartość 0 oznacza nieskończony transfer, bez ograniczeń. |
| CY\_FX\_SLFIFO\_DMA\_RX\_SIZE | 0 |
| CY\_FX\_SLFIFO\_THREAD\_STACK | 0x04 | Wielkość stosu wątku |
| CY\_FX\_SLFIFO\_THREAD\_PRIORITY | 8 | Priorytet wątku |
| CY\_FX\_EP\_PRODUCER | 0x01 | Numer punktu końcowego (ang. endpoint) |
| CY\_FX\_EP\_CONSUMER | 0x81 |
| CY\_FX\_PRODUCER\_USB\_SOCKET | CY\_U3P\_UIB\_SOCKET\_PROD\_1 | Nazwa gniazda USB odpowiadającego za źródło danych |
| CY\_FX\_CONSUMER\_USB\_SOCKET | CY\_U3P\_UIB\_SOCKET\_CONS\_1 | Nazwa gniazda USB odpowiadającego za pobieranie danych |
| CY\_FX\_PRODUCER\_PPORT\_SOCKET | CY\_U3P\_PIB\_SOCKET\_0 | Nazwa gniazda GPIF odpowiadającego za źródło danych |
| CY\_FX\_CONSUMER\_PPORT\_SOCKET | CY\_U3P\_PIB\_SOCKET\_3 | Nazwa gniazda GPIF odpowiadającego za pobieranie danych |

Poniżej wyszczególniono spis modyfikacji przeprowadzonych w głównym kodzie źródłowym (plik *cyfxslfifosync.c*):

* dołączono plik *cyfxgpif2config.h* dyrektywą preprocesora #*include*,
* zadeklarowano zmienną *burst\_length*, która odpowiada za liczbę pakietów podczas jednego transferu oraz przypisano jej odpowiednie wartości dla różnych prędkości USB. Następnie zmienną wykorzystano w przypisaniu wartości jednemu z elementów struktury *CyU3PEpConfig\_t*, która stanowi drugi argument funkcji *CyU3PSetEpConfig* do konfiguracji punktu końcowego (ang. endpoint),
* utworzono automatyczny i manualny kanał DMA, wykorzystano stałe z pliku nagłówkowego oraz funkcję *CyU3PDmaChannelCreate*,
* utworzono funkcję GPIFErrorCallback, który służy do debugowania programu, komunikaty do komputera wysyłane są przy użyciu interfejsu UART ,
* użyto funkcji *CyU3PGpifLoad* oraz danych wygenerowanych przez narzędzie GPIF II Designer, aby skonfigurować programowalną maszynę stanów,
* wybrano, skonfigurowano oraz następnie dokonano aktywacji gniazda GPIF, użyto funkcji *CyU3PGpifSocketConfigure* (w przypadku, gdy wcześniej skonfigurowano flagi częściowe),
* uruchomiono programowalną maszynę stanów GPIF dzięki funkcji *CyU3PGpifSMStart*, jako argumenty przyjmuje ona stan początkowy maszyny stanów,
* umożliwiono rejestrację powiadomień występujących podczas przerwań procesora za pomocą funkcji *CyU3PPibRegisterCallback*,
* ustawiono pętlę PLL, dzięki której doprowadzono do FX3 sygnał zegarowy o częstotliwości 400 MHz, odpowiedzialna linia kodu: *clockConfig.setSysClk400 = CyTrue*,
* skonfigurowano funkcję *CyU3PDeviceCacheControl*, aby pamięć podręczna była czyszczona przez użytkownika.

### 4.3.3. Tryb transmisji danych w obu kierunkach

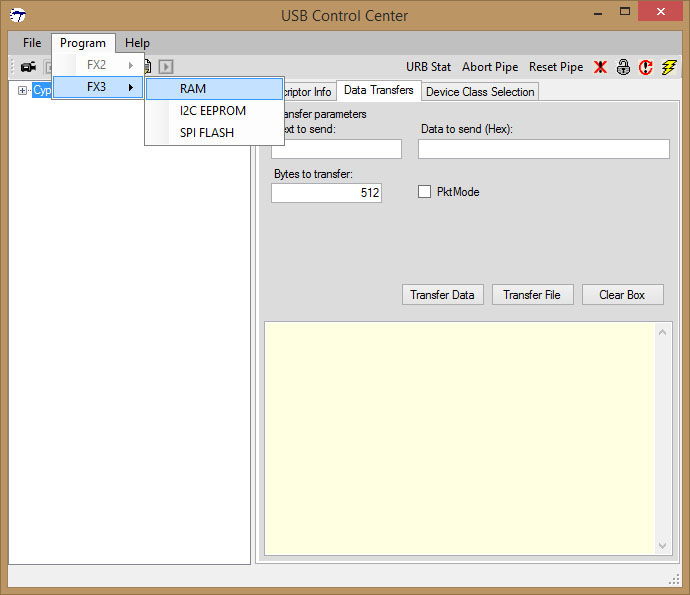
Projekt realizujący transfer danych w obu kierunkach nie różni się diametralnie od programu opisanego w poprzednim podrozdziale (większość kodu pozostała identyczna). Za bazę posłużył przykładowy projekt przygotowany przez firmę Cypress. Zmiany zostały przeprowadzone w następujących miejscach:

* zmieniono wartość stałej *DMA\_BUF\_SIZE* z 16 na 1,
* ustawiono stałą wartość zmiennej *burst\_length,*
* zmieniono wartości stałych *CY\_FX\_SLFIFO\_DMA\_BUF\_COUNT\_P\_2\_U* oraz *CY\_FX\_SLFIFO\_DMA\_BUF\_COUNT\_U\_2\_P* z 8 i 4 na 2,
* przypisano wartość "1" zmiennej globalnej *glIsApplnActive* odpowiadającej za uruchomienie trybu transferu danych w obu kierunkach (w przypadku strumieniowego transferu danych zmienna posiadała wartość "0").

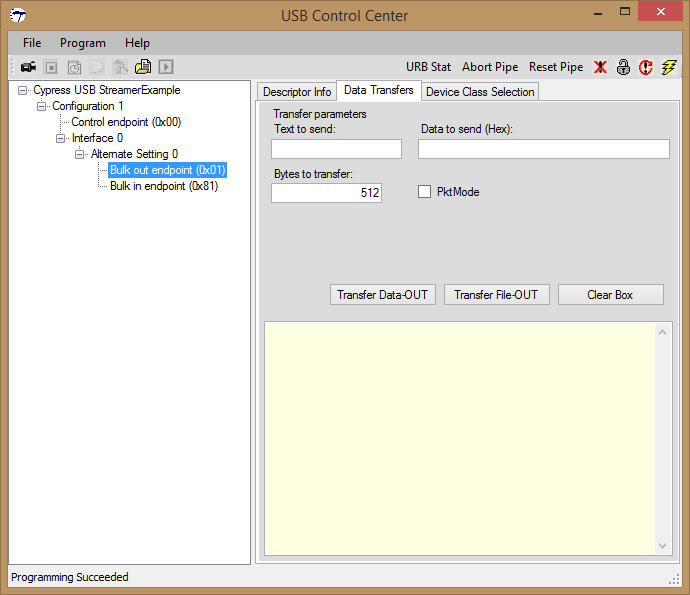
## 4.4. USB Control Center - programowanie układu

Po kompilacji projektu otrzymano plik o rozszerzeniu .img, który przesłano do pamięci RAM FX3 w celu zaprogramowania układu [6].

Uruchomiono okno programu USB Control Center. Po zaznaczeniu w lewej części okna urządzenia *Cypress USB BootLoader* wybrano z paska narzędzi opcję *Program - FX3 - RAM* (Rys. 16). Należało wskazać plik .img na dysku komputera, po czym nastąpiło wgranie programu do pamięci procesora. Zauważono zmianę nazwy urządzenia na *Cypress USB Streamer*, po czym rozwinięto drzewo konfiguracji oraz uzyskano możliwość wysyłania lub odbierania danych w konkretnych punktach końcowych (Rys. 17).



Rys. . Wybór pliku .img do zaprogramowania układu



Rys. . Po zaprogramowaniu FX3 uzyskano dostęp do punktów końcowych

# 5. Xilinx Spartan 3E

Spartan 3E to rodzina układów programowalnych FPGA firmy Xilinx, która została zaprojektowana w celu zaspokojenia rynku układów o bardzo dużej pojemności w przystępnej cenie. W tej pracy wykorzystano układ XC3S500E, który posiada 10476 programowalnych bramek. Największy układ z tej rodziny posiada ich 33192, a najmniejszy 2160. XC3S500E klasyfikuje się w środkowym miejscu tabeli biorąc pod uwagę zaawansowanie układu [15]. Poniżej przedstawiono najważniejsze cechy układu oraz zestawu ewaluacyjnego [12]:

* standard logiczny: 3,3 V,
* moduł Cyfrowego Zarządzania Sygnałem Zegarowym - 4 bloki,
* maksymalna ilość wejść/wyjść - 232,
* możliwość konfiguracji po załączeniu zasilania za pomocą Xilinx 4 Mbit Flash PROM,
* zewnętrzny zegar o częstotliwości 50 MHz,
* złącze Hirose FX2 do szybkiej transmisji danych,
* cztery przełączniki wykorzystywane do zmiany trybu transmisji,
* wyświetlacz LCD ze sterownikiem HD44780 pracujący w trybie 4-bitowym.

## 5.1. Środowisko projektowe

Do syntezy i implementacji projektu wykorzystano pakiet WebPack ISE 14, który jest dostępny do pobrania za darmo ze strony producenta. Posłużono się najnowszą wersją środowiska programistycznego udostępnioną przez firmę Xilinx. WebPack ISE to potężny pakiet narzędziowy, który standardowo zawiera wszystkie niezbędne moduły, między innymi: syntezer języka HDL, PlanAhead (planowanie, rozkład i dostosowanie opcji wyprowadzeń wejściowych/wyjściowych) czy Xilinx Impact (programowanie układu docelowego lub układu wykorzystującego plik PROM do wstępnej konfiguracji FPGA).

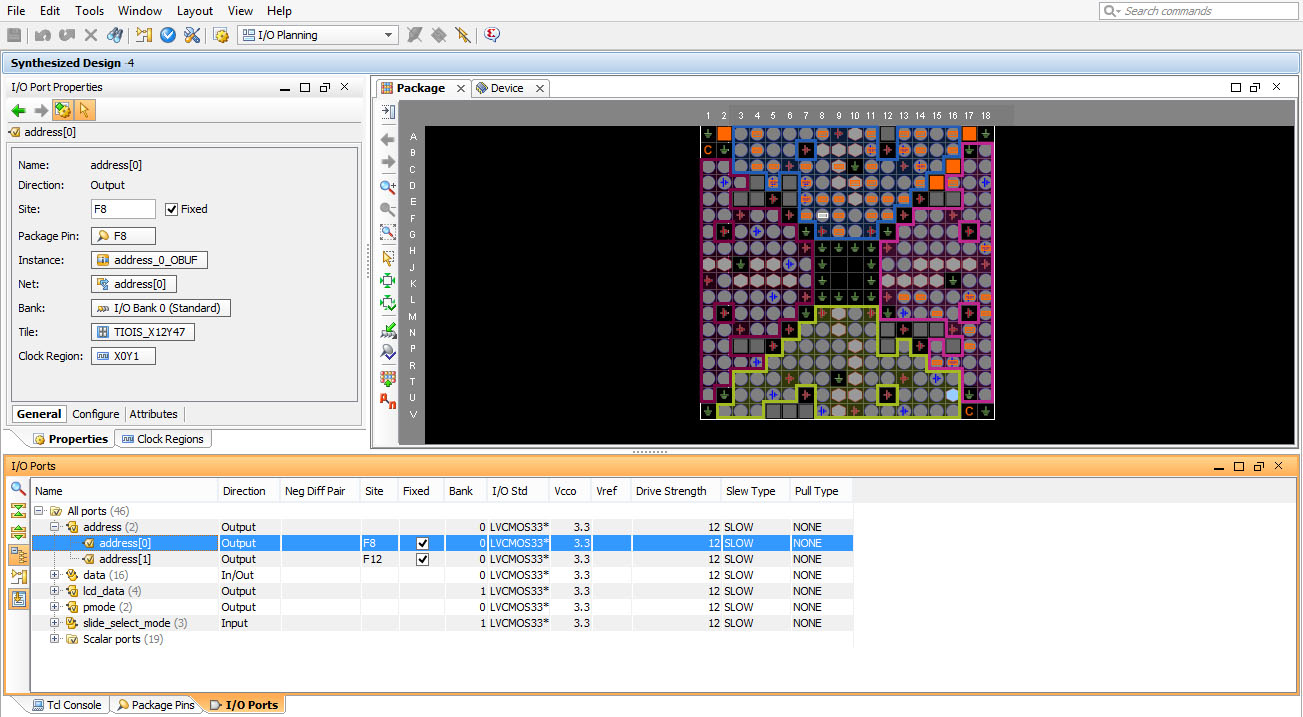
## 5.2. Realizacja projektu i przypisanie wyprowadzeń

Utworzono nowy projekt wybierając opcję w menu programu *File - New Project*. Podano nazwę projektu *Slave Fifo* oraz wybrano miejsce na dysku komputera w celu ulokowania plików projektu. W opcjach szczegółowych wybrano z rozwijanej listy następujące parametry układu:

* Rodzina: Spartan 3E,
* Układ: XC3S500E,
* Obudowa: FG320,
* Prędkość: 4,
* Synteza: XST (VHDL/Verilog),
* Język programowania: VHDL.

Następnie dodano do projektu pliki źródłowe z kodem napisanym w języku VHDL klikając prawym klawiszem myszy na opcję *New source* widoczną w drzewie projektu.

Kolejnym ważnym krokiem w konfiguracji projektu było utworzenie pliku o rozszerzeniu .ucf, który zawiera przypisania wyprowadzeń do zadeklarowanych wyprowadzeń układu. Posłużono się modułem Xilinx PlanAhead (Rys. 18).



Rys. 18. Widok programu Xilinx PlanAhead

W części *I/O Ports* znalazły się wszystkie wejścia i wyjścia zdefiniowane w części deklaracyjnej projektu (ang. entity). Po wyborze konkretnego sygnału w zakładce *I/O Port Properties* dokonano przypisania do odpowiedniego wyprowadzenia FPGA (posłużono się dokumentacją zestawu [12]). Zaznaczono wszystkie sygnały (*All ports*) i następnie wybrano opcję *Configure I/O Ports*. Zmieniono domyślny standard z LVCMOS18 na LVCMOS33 (do portów doprowadzono znamionowe napięcie 3,3 V) i moc sygnału z 4 na 12. Ustawiono zwiększoną prędkość (ang. fast) jedynie wyprowadzeniom dostępnym na złączu Hirose FX2 (sygnały wykorzystywane do komunikacji z EZ-USB FX3), reszta portów pozostała w standardowym wolnym trybie (ang. slow). Po zapisaniu ustawień moduł automatycznie wygenerował plik .ucf, który był niezbędny do syntezy projektu.

Poniżej zamieszczono Tab. 8 z przypisaniami każdego wyprowadzenia wraz z krótkim opisem.

Tab. . Przypisania wyprowadzeń FPGA

|  |  |  |
| --- | --- | --- |
| **Nazwa sygnału** | **Przypisanie** | **Opis** |
| clock50 | C9 | Wewnętrzny zegar 50 MHz |
| reset\_from\_slide | N17 | Reset układu |
| slide\_select\_mode[0] | L13 | Wybór trybu transmisji |
| slide\_select\_mode[1] | L14 |
| slide\_select\_mode[2] | H18 |
| led\_buffer\_empty\_show | F9 | Status pamięci FIFO |
| lcd\_e | M18 | Sygnał Enable LCD |
| lcd\_rs | L18 | Sygnał Register Select LCD |
| lcd\_rw | L17 | Sygnał Read/Write LCD |
| lcd\_srataflash\_disable | D16 | Wyłączenie modułu FLASH LCD |
| lcd\_data[0] | R15 | 4-bitowa linia danych |
| lcd\_data[1] | R16 |
| lcd\_data[2] | P17 |
| lcd\_data[3] | M15 |
| clock100\_out | D10 | Sygnał zegarowy do FX3 |
| flaga | A13 | Flaga informująca o stanie buforu FX3 |
| flagb | C5 |
| flagc | E7 |
| flagd | F7 |
| pktend | D5 | Informacja o krótkim pakiecie |
| sloe | C7 | Sygnał Output/Enable FX3 |
| slwr | D7 | Sygnał Write FX3 |
| slcs | A4 | Sygnał Chip Select FX3 |
| slrd | B4 | Sygnał Read FX3 |
| address[0] | F8 | Adresowanie wątków FX3 |
| address[1] | F12 |
| data[0] | A14 | Linia danych |
| data[1] | C14 |
| data[2] | D14 |
| data[3] | B14 |
| data[4] | E11 |
| data[5] | A16 |
| data[6] | E9 |
| data[7] | C11 |
| data[8] | B13 |
| data[9] | G9 |
| data[10] | B16 |
| data[11] | B11 |
| data[12] | C4 |
| data[13] | A11 |
| data[14] | E13 |
| data[15] | A8 |
| pmode[0] | D11 | Sygnały do debugowania |
| pmode[1] | F11 |
| reset\_to\_fx3 | E8 | Sygnał resetujący FX3z FPGA |
| reset\_from\_fx3 | E12 | Sygnał resetujący FPGA z FX3 |

## 5.3. Moduł Cyfrowego Zarządzania Sygnałem Zegarowym (DCM - Digital Clock Manager)

Zestaw Spartan 3E został ustawiony jako układ główny (ang. master) w interfejsie synchronicznej kolejki FIFO, więc kontroler EZ-USB FX3 skonfigurowano jako układ podrzędny (ang. slave). Zastosowano się do wytycznych przygotowanych przez firmę Cypress, aby zminimalizować ryzyko powstawania błędów podczas transferu danych.

* skonfigurowano pętlę PLL w zestawie EZ-USB FX3, która przekształca doprowadzony sygnał zegarowy o częstotliwości 19,2 MHz do zalecanej częstotliwości 400 MHz,
* doprowadzono do programowalnej maszyny stanów GPIF II sygnał zegarowy pochodzący z układu nadrzędnego (FPGA) o częstotliwości 100 MHz.

Pierwszy podpunkt zrealizowano za pomocą kilku instrukcji wykorzystując API Cypressa (problem omówiono szerzej w podrozdziale 4.3). Uzyskanie sygnału zegarowego o częstotliwości 100 MHz z 50 MHz (nominalna częstotliwość zegara umieszczonego na płytce uruchomieniowej Spartan 3E) wiązała się z wykorzystaniem modułu zaimplementowanego przez firmę Xilinx, a mianowicie DCM (moduł Cyfrowego Zarządzania Sygnałem Zegarowym, ang. Digital Clock Manager).

Pojedynczy blok DCM składa się z 4 części: syntezera częstotliwości DFS (ang. Digital Frequency Synthesizer), pętli DLL (ang. Delay Locked Loop), programowanego przesuwania fazy (ang. Phase Shifter) oraz zespołu logiki (ang. Status Logic) [13]. Moduł zapewnia integrację zaawansowanych technik taktowania układu programowalnego. Co za tym idzie, boki DCM w rodzinie Spartan 3 rozwiązują wiele problemów związanych zwłaszcza z uzyskiwaniem dużych częstotliwości. Możliwości oferowane przez DCM:

* mnożenie lub dzielenie doprowadzonego sygnału zegarowego - synteza kompletnie nowej wartości częstotliwości,
* poprawa zbocza sygnału, eliminacja efektu propagacji,
* kontrola 50% wypełnienia,
* przesuwanie fazy sygnału.

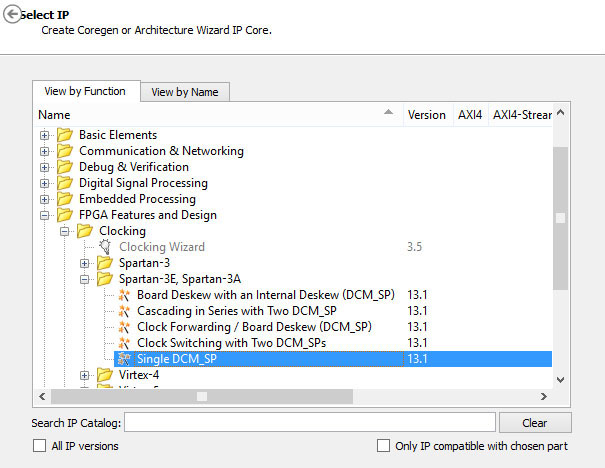
Celem wykorzystania modułu DCM było uzyskanie częstotliwości 100 MHz, a zatem częstotliwości dwukrotnie większej od oryginalnej. Nie skorzystano zatem z możliwości wyboru mnożników i dzielników, zaznaczono opcję *CLK2FX*. Jest to rozwiązanie znacznie mniej skomplikowane. Poniżej przedstawiono proces konfiguracji modułu DCM.

* dodano nowe źródło w drzewie projektu,
* wybrano opcję IP (*CORE Generator & Architecture Wizard*),
* rozwinięto drzewo opcji: *FPGA Features and Design* - *Clocking* - *Spartan 3E* - *Single DCM\_SP* (Rys. 19),
* podano wartość 50 MHz w polu *Input Clock Frequency*, jako źródło sygnału zegarowego wybrano zegar zewnętrzny (*CLKIN Source - External*),
* zaznaczono opcję *CLK2X* oraz *RST* (Rys. 20).

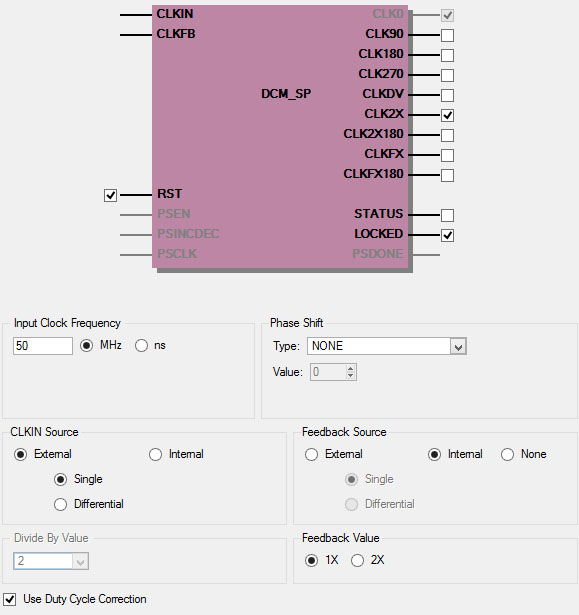
Po zatwierdzeniu ustawień i wygenerowaniu modułu przez IDE wybrano opcję *View HDL Instantiation Template* w celu uzyskania sekcji *component* (deklaracja jednostki projektowej) oraz *port map* (mapowanie sygnałów). Dołączono je do głównego kodu źródłowego. W sekcji mapowania dokonano przypisania wchodzących oraz wychodzących sygnałów do modułu (Tab. 9).

Tab. . Opis sygnałów modułu DCM

|  |  |  |
| --- | --- | --- |
| **Nazwa sygnału** | **Typ** | **Opis sygnału** |
| CLKIN\_IN | wejście | zewnętrzny zegar 50 MHz |
| RST\_IN | wejście | resetowanie modułu |
| CLKIN\_IBUFG\_OUT | wyjście | globalny bufor |
| CLK0\_OUT | wyjście | 50 MHz po korekcji zbocza |
| CLK2X\_OUT | wyjście | 100 MHz po korekcji zbocza |



Rys. 19. Wybór DCM z listy IP Cores



Rys. 20. Konfiguracja bloku DCM

## 5.4. Pamięć FIFO

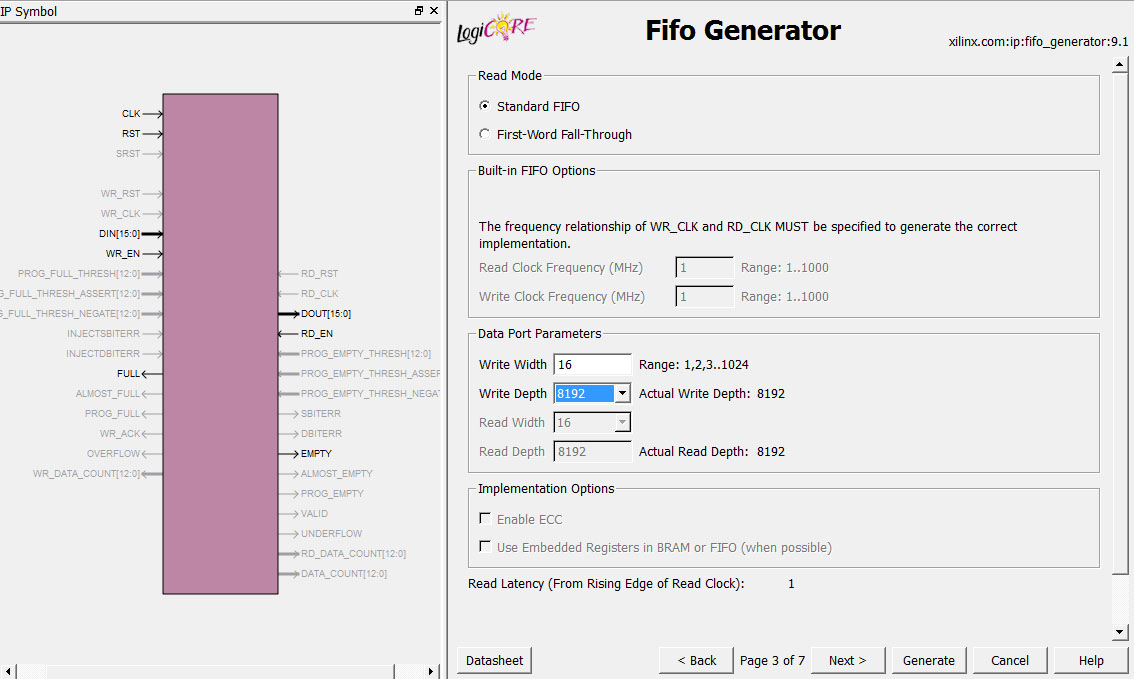
W celu implementacji transferu danych w obu kierunkach (ang. Loopback Transfer, czyli wysyłanie i odbiór identycznych pakietów danych w komputerze) niezbędne było użycie pamięci FIFO (ang. First In, First Out) [14]. Posłużono się modułem *Fifo Generator* zaimplementowanym przez firmę Xilinx dostępnym (podobnie jak DCM) w katalogu *IP Core*. Konfiguracja (Rys. 21) przebiegła następująco:

* dodano nowe źródło w drzewie projektu, wybrano *IP Cores* oraz podano nazwę modułu,
* rozwinięto listę *Memories & Storage Elements - Fifo Generator* w wersji 9.1,
* ustawiono, aby port wejściowy i wyjściowy posiadały identyczną pojemność: szerokość 16 bitów, głębokość 8192 bitów.

Po zatwierdzeniu ustawień postąpiono analogicznie jak w przypadku DCM-a, czyli skopiowano do pliku źródłowego kod wygenerowany w IDE. W Tab. 10 zawarto spis oraz krótki opis poszczególnych sygnałów.

Tab. . Opis sygnałów modułu pamięci FIFO

|  |  |  |
| --- | --- | --- |
| **Nazwa sygnału** | **Typ** | **Opis sygnału** |
| clk | wejście | zegar 100 MHz |
| rst | wejście | resetowanie pamięci |
| din | wejście | 16-bitowe wejście danych |
| wr\_en | wejście | pozwolenie na zapis danych |
| rd\_en | wejście | pozwolenie na odczyt danych |
| dout | wyjście | 16-bitowe wyjście danych |
| full | wyjście | sygnalizacja pełnej pamięci |
| empty | wyjście | sygnalizacja pustej pamięci |



Rys. 21. Konfiguracja pamięci FIFO

## 5.5. Struktura programu dla FPGA

Program napisano w języku opisu sprzętu VHDL. Został on podzielony na indywidualne moduły. Każdy moduł działa na zasadzie maszyny o skończonej liczbie stanów [9]:

* slave\_fifo\_main - moduł główny, zarządzanie sygnałami,
* lcd\_controller - obsługa wyświetlacza LCD,
* slave\_fifo\_stream\_write\_to\_fx3 - strumieniowy transfer danych z FPGA do PC,
* slave\_fifo\_stream\_read\_from\_fx3 - strumieniowy transfer danych z PC do FPGA,
* slave\_fifo\_loopback - transfer danych w obu kierunkach.

Wybór trybu transmisji danych umożliwiają trzy przełączniki oznaczone jako SW0, SW1 oraz SW2 na zestawie uruchomieniowym. Czwarty przełącznik (SW3) został tak skonfigurowany, aby móc przywrócić stan początkowy programu (reset). Poniżej przedstawiono tabelę z możliwymi ustawieniami przełączników (Tab. 11).

Tab. . Wybór trybu działania przez przełączniki

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Stan** | **Przełącznik SW3** | **Przełącznik SW2** | **Przełącznik SW1** | **Przełącznik SW0** | **Opis działania** |
| Niski | Niski | Niski | Niski | Stan bezczynności |
| Niski | Niski | Niski | Wysoki | Pętla |
| Niski | Niski | Wysoki | Niski | Ciągłe wysyłanie danych z PC do FPGA |
| Niski | Wysoki | Niski | Niski | Ciągłe wysyłanie danych z FPGA do PC |
| Wysoki | Niski | Niski | Niski | Reset |
| Wysoki | Wysoki | Dowolny | Dowolny | Reset |
| Niski | Wysoki | Wysoki | Dowolny | Stan bezczynności |

Skonfigurowano wyświetlacz LCD, aby przedstawiał informację o aktualnym trybie. Ponadto dla trybów związanych z transferem danych druga linia wyświetlacza prezentuje aktualnie pobierane lub wysyłane dane (16 bitów). Kolejne podrozdziały przedstawiają sposób działania każdego modułu.

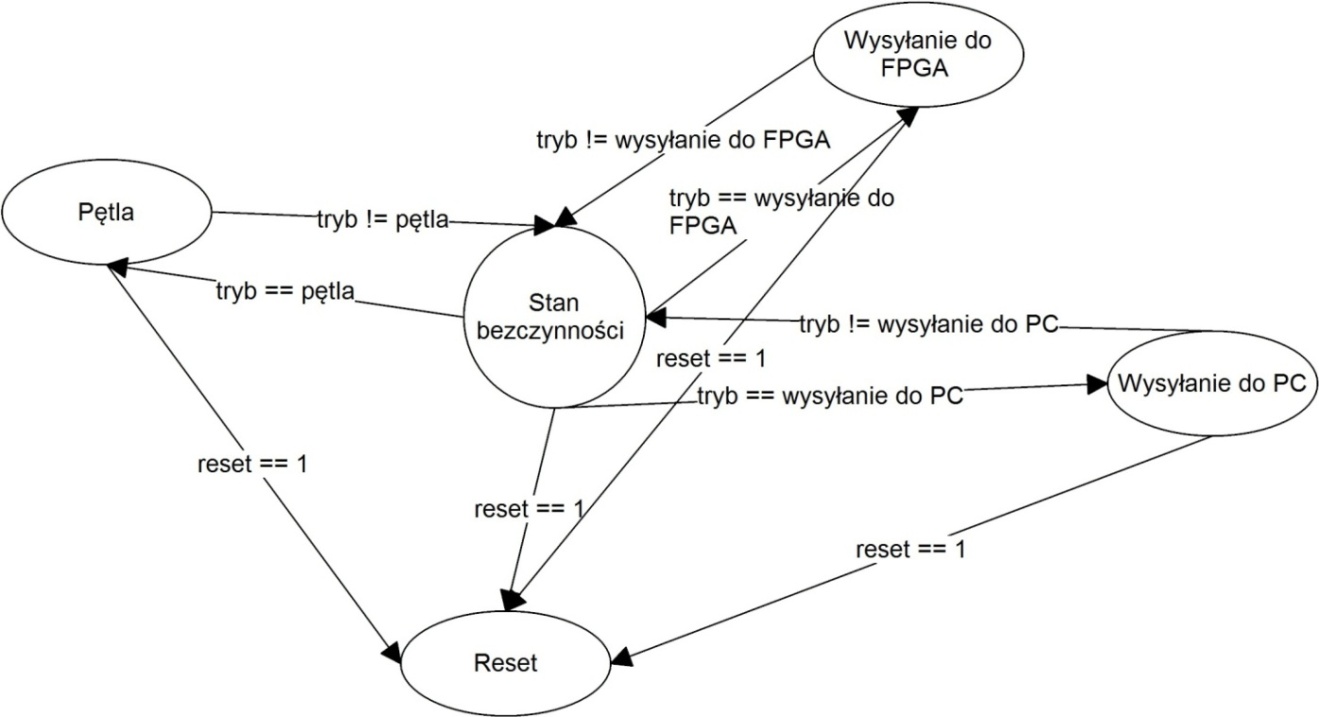
### 5.5.1. Główny program - Slave Fifo Main

Główny program zawiera deklarację jednostki (ang. entity), w której zdefiniowane zostały wszystkie sygnały wyjściowe i wejściowe (opisane wcześniej w Tab. 8). Natomiast w architekturze zadeklarowano stałe, pomocnicze sygnały oraz zmienne maszyny stanów. Następnie zmapowano wszystkie komponenty. Napisano funkcję *vector\_to\_string*, która ma za zadanie konwertować zmienne typu *std\_logic\_vector* na typ *string*, później wykorzystywane do prezentowania danych na ekranie LCD.

W sekcji głównej programu znalazły się procesy, które w zależności od wybranej transmisji danych manipulują stanami głównych sygnałów, tzn. sloe, slrd, slcs czy sloe (ich znaczenie zostało omówione w rozdziale trzecim). Pozostałe procesy mają za zadanie realizować transfer danych w obu kierunkach oraz monitorować stan flag A, B, C i D. Ponadto w programie umieszczono główną maszynę stanów odpowiadającą za przechodzenie do wybranego trybu transmisji. Na schemacie przedstawiono zasadę działania głównego programu (Rys. 22).

Stany (zmienne) głównego programu:

* idle\_state (stan bezczynności),
* loopback\_state (tryb transferu danych w obu kierunkach),
* stream\_read\_from\_fx3\_state (tryb strumieniowego transferu danych do FPGA),
* stream\_write\_to\_fx3\_state (tryb strumieniowego transferu danych do komputera).



Rys. 22. Maszyna stanów głównego programu

Analizując powyższy diagram można dojść do wniosku, że nie jest możliwe przejście z trybu np. strumieniowego transferu danych z PC do FPGA do trybu transferu danych w obu kierunkach pomijając stan bezczynności (idle). Adres aktualnego wątku został uzależniony od trybu, w którym znajduje się maszyna stanów. Jedynym wyjątkiem jest przejście do stanu resetu, przebiega ono asynchronicznie i niezależnie od obecnego trybu pracy FPGA.

W programie głównym umieszczono jeszcze obsługę ekranu LCD, która została również podzielona na poszczególne etapy. Maszyna stanów składa się z:

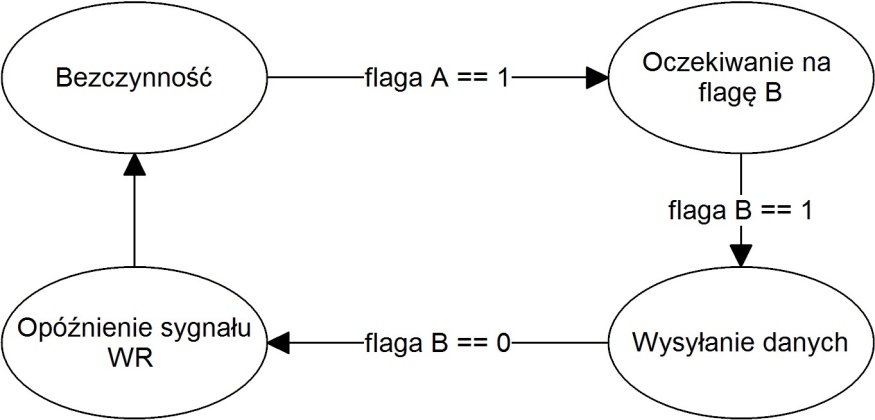
* idle (stan bezczynności),
* start (wyzerowanie licznika),
* clearscr (wyczyszczenie ekranu z poprzedniego napisu oraz wyzerowanie licznika),
* move1 (przejście do 1 miejsca w 1 linii),
* send1 (wysłanie 2 bajtów danych),
* move2 (przejście do miejsca w 2 linii),
* send2 (wysłanie 2 bajtów danych).

Moduł *lcd\_controller* jest odpowiedzialny za obsługę 4-bitowej linii danych oraz pozostałych sygnałów ekranu LCD.

### 5.5.2. Transmisja strumienia danych do komputera - Stream Write to FX3

Po deklaracji jednostki projektowej, stałych i sygnałach pomocniczych umieszczono deklarację maszyny stanów odpowiedzialną za poprawne działanie trybu transmisji strumienia danych do komputera. Na poniższym diagramie zaprezentowano zasadę działania maszyny stanów (Rys. 23).

* idle,
* wait\_flagb,
* write,
* wr\_delay.



Rys. 23. Schemat działania transferu strumienia danych do komputera

Maszyna stanów jest odpowiedzialna za monitorowanie sygnałów flag pochodzących z kontrolera USB (wskazują dostępność buforów DMA) oraz zmienia ona stany logiczne sygnałów związanych z transmisją danych.

Omawiany tryb transmisji manipuluje jedynie stanem slwr, czyli sygnałem odpowiedzialnym za zapis danych.

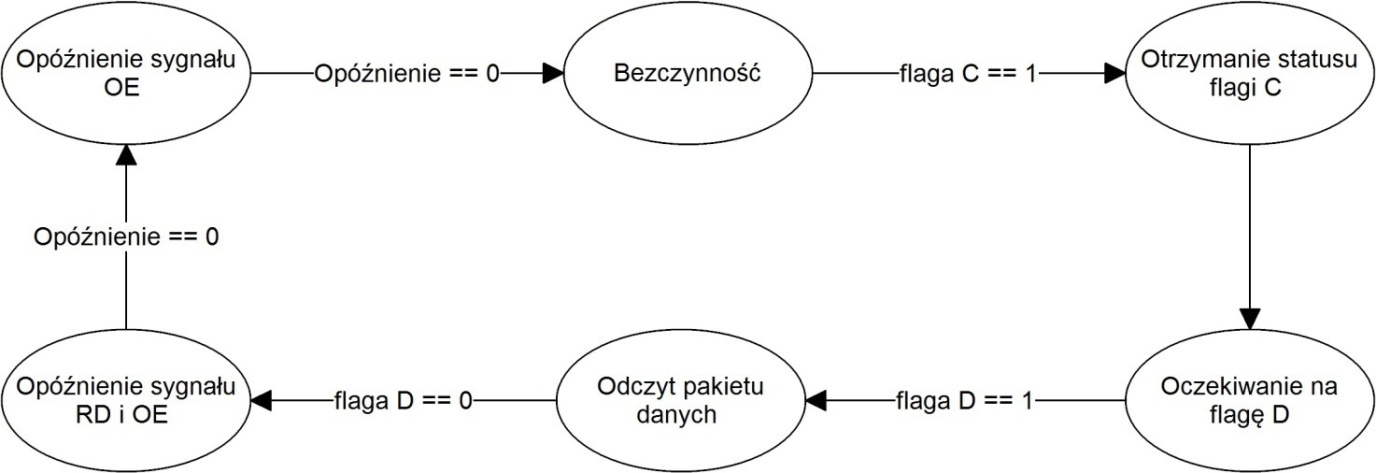
* slcs = 0 (wartość stała),
* sloe = 1 (wartość stała),
* slrd = 1 (wartość stała),
* slwr = 0 jeśli maszyna stanów przeszła do stanu *stream\_in\_write* oraz stan flagi B wynosi 1, w innych przypadkach slwr = 1.

Postanowiono, że program będzie wysyłał do komputera identyczny pakiet danych zapisany w dwójkowym systemie liczbowym: 0101011101001101. Po przekształceniu na system heksadecymalny otrzymano 0x574D. Liczba 0x57 odpowiada literze "W", a 0x4D literze "M" w kodzie ASCII.

### 5.5.3. Odbiór strumienia danych z komputera - Stream Read from FX3

Dla trybu odbioru strumienia danych z komputera zdefiniowano 6 stanów. Poniżej wyszczególniono w podpunktach kolejne etapy, natomiast mechanizm działania maszyny stanów został przedstawiony na Rys. 24.

* idle,
* flagc\_rcvd,
* wait\_flagd,
* read,
* read\_rd\_oe\_delay,
* read\_oe\_delay.



Rys. . Schemat działania transferu strumienia danych do FPGA

Odczytany pakiet danych jest wyświetlany w czasie rzeczywistym na wyświetlaczu LCD, jednak nie jest on nigdzie zapisywany (pamięć FIFO została użyta jedynie w trybie dwukierunkowej transmisji danych).

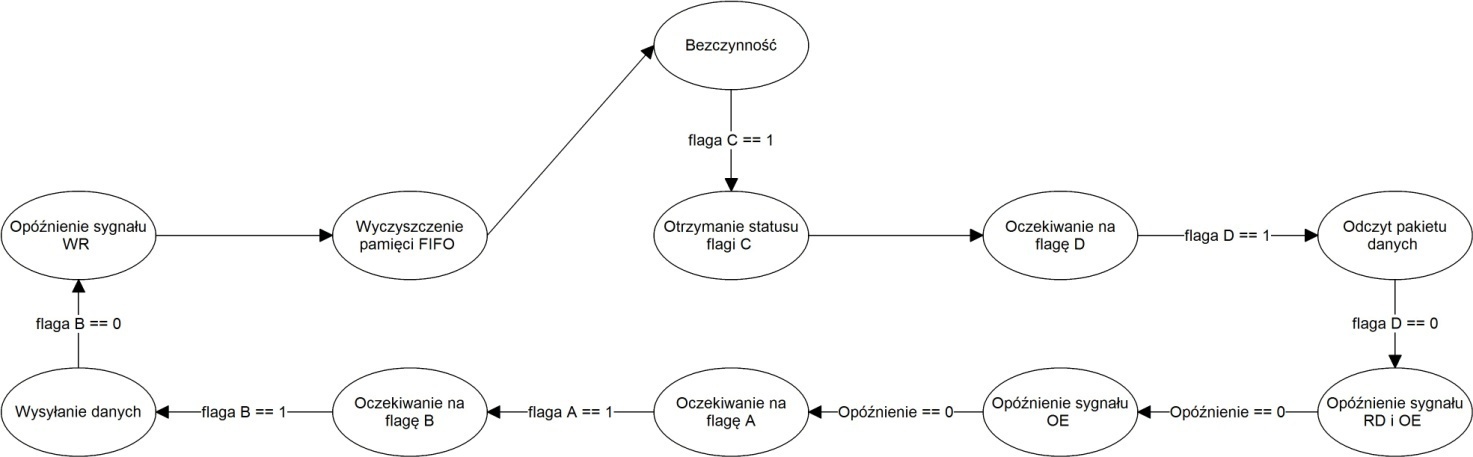
Na osoby akapit zasługuje omówienie opóźnień generowanych po odczycie danych. Zgodnie z zasadami (opisanymi w dokumentacji firmy Cypress odnośnie interfejsu synchronicznej kolejki FIFO [9]) w przypadku użycia częściowych flag jest konieczne, aby FX3 monitorował sygnał *SLRD* jeszcze przez kolejne 3 takty zegara (oe\_delay) po standardowym opóźnieniu 2 cykli zegara (rd\_oe\_delay).

Maszyna stanów w omawianym trybie modyfikuje stany logiczne dwóch sygnałów odpowiedzialnych za zapis, są to:

* slcs = 0 (wartość stała),
* slwr = 1 (wartość stała),
* sloe = 0 jeśli maszyna stanów znalazła się w stanie *read* oraz podczas obu opóźnień, w pozostałych przypadkach sloe = 1,
* slrd = 0 jeśli maszyna stanów znalazła się w stanie *read* oraz podczas pierwszego opóźnienia *rd\_oe\_delay*, w pozostałych przypadkach slrd = 1.

### 5.5.4. Transmisja danych w obu kierunkach - Loopback Transfer

Ostatni z zaimplementowanych trybów transferu danych jako jedyny wykorzystuje (omówioną w rozdziale 5.4) pamięć FIFO. Pakiet danych odebrany z magistrali USB jest zapisywany w pamięci FIFO, a następnie przy zwolnieniu się miejsca w kanale DMA następuje pobranie tego pakietu z FIFO, wyczyszczenie pamięci oraz odesłanie danych z powrotem do komputera. Tryb powstał z połączenia dwóch poprzednich transmisji, zatem utworzono najbardziej rozbudowany diagram (Rys. 25).



Rys. . Schemat działania transferu danych w obu kierunkach

Maszyna stanów w kodzie VHDL zawiera wymienione poniżej elementy:

* idle,
* flagc\_rcvd,
* wait\_flagd,
* read,
* read\_rd\_oe\_delay,
* read\_oe\_delay,
* wait\_flaga,
* wait\_flagb,
* write,
* write\_wr\_delay,
* flush\_fifo.

W tym przypadku maszyna stanów monitoruje wszystkie cztery flagi oraz manipuluje stanami trzech sygnałów:

* slcs = 0 (wartość stała),
* slwr = 0 jeśli maszyna stanów znalazła się w stanie *stream\_in\_write* oraz stan flagi B wynosi 1, w innych przypadkach slwr = 1,
* sloe = 0 jeśli maszyna stanów znalazła się w stanie *read* oraz podczas obu opóźnień, w pozostałych przypadkach sloe = 1,
* slrd = 0 jeśli maszyna stanów znalazła się w stanie *read* oraz podczas pierwszego opóźnienia *rd\_oe\_delay*, w pozostałych przypadkach slrd = 1.

## 5.6. Programowanie FPGA Spartan 3E

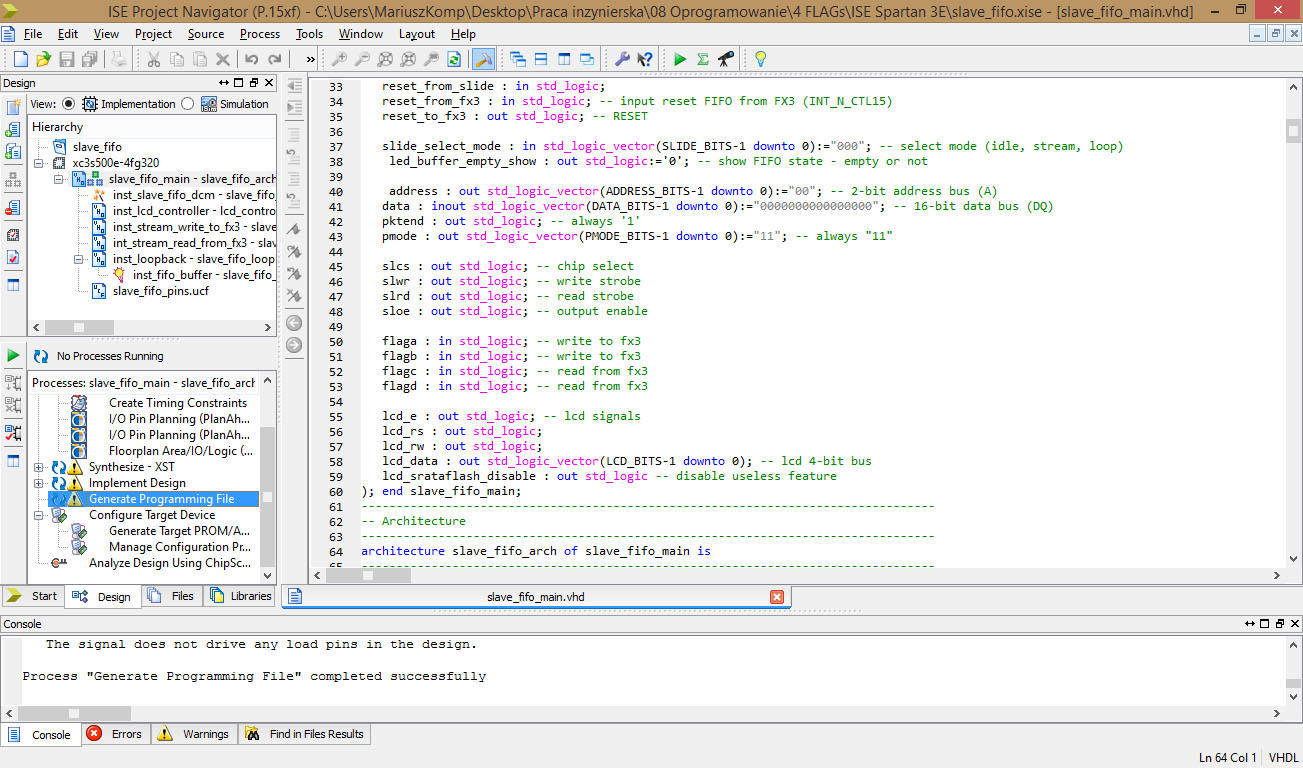
Zestaw ewaluacyjny Spartan 3E wspiera cztery możliwości zaprogramowania układu programowalnego [12]:

* bezpośrednia konfiguracja przy wykorzystaniu interfejsu JTAG i wbudowanego na płytce konwertera USB (ang. In System Programming),
* zaprogramowanie układu Xilinx XCF04S wygenerowanym uprzednio plikiem Flash PROM, który konfiguruje układ bezpośrednio po załączeniu zasilania,
* zaprogramowanie układu ST Microelectronics przy wykorzystaniu magistrali SPI, która następnie konfiguruje układ bezpośrednio po załączeniu zasilania,
* zaprogramowanie pamięci Intel SrataFlash plikiem Flash PROM, a następnie konfiguracja FPGA przy wykorzystaniu dwóch trybów: *BPI Up* oraz *BPI Down*.

Podczas testowania interfejsu posłużono się pierwszą metodą, ponieważ jest ona najszybsza ze wszystkich (wystarczy jedno polecenie, aby wgrać program do układu). Z kolei drugą możliwość wykorzystano, gdy program został ukończony. Spartan 3E podczas załączenia zasilania pobiera konfigurację z układu XCF04S, nie zachodzi potrzeba jakiejkolwiek dalszej ingerencji użytkownika.

### 5.6.1. Konfiguracja z wykorzystaniem interfejsu JTAG

Dokonano syntezy projektu oraz wygenerowano plik do zaprogramowania układu. W tym celu wybrano opcję *Generate Programming File* widoczną pod drzewem projektu. W konsoli ukazał się komunikat *Process "Generate Programming File" completed successfully*, który poinformował o poprawnie przeprowadzonej syntezie i kompilacji projektu (Rys. 26).



Rys. . Komunikat w konsoli informujący o poprawnie przeprowadzonej syntezie i kompilacji projektu

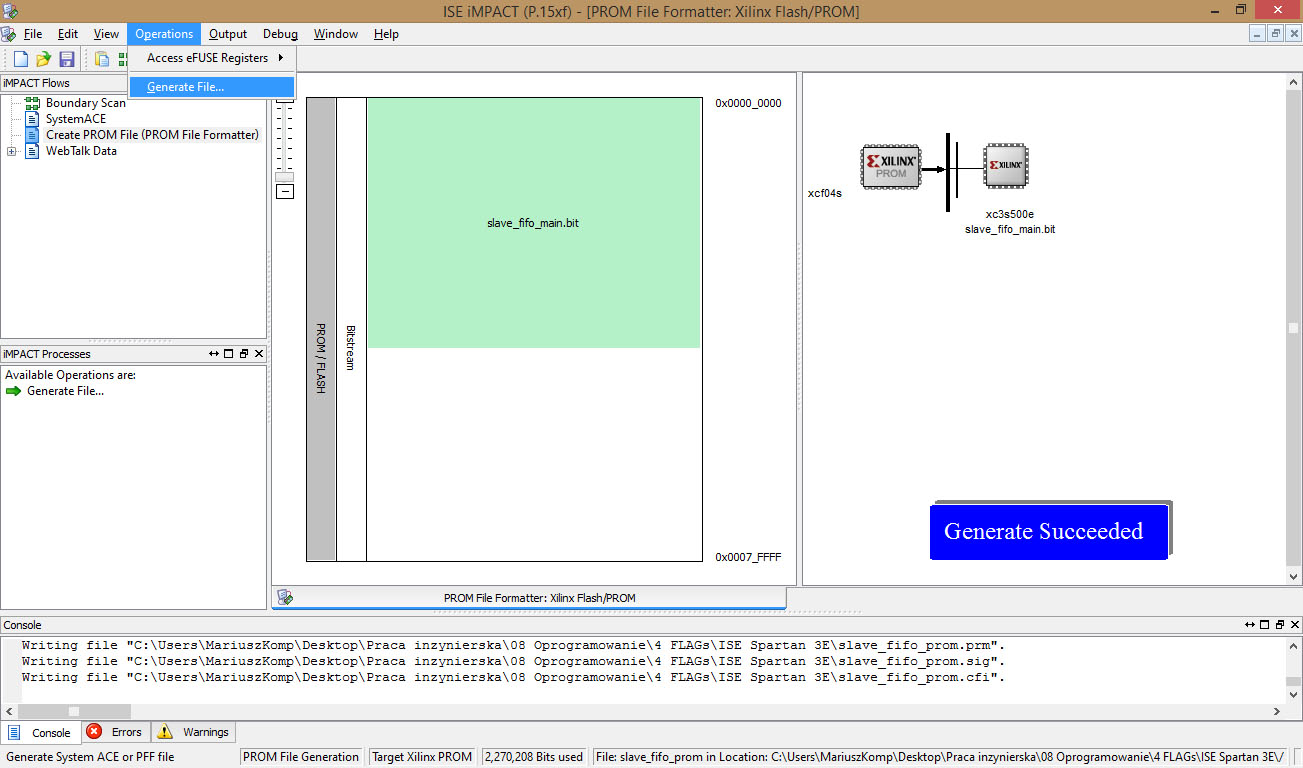
Po wygenerowaniu pliku wybrano opcję *Configure Target Device*. Uruchomiono program Impact ISE i wybrano opcję *Boundary Scan*. Następnie w białym polu okna naciśnięto prawy przycisk myszy i wybrano z menu opcję *Initialize Chain*. Wskazano plik slave\_fifo.bit, który wygenerowano w poprzednich krokach. Pominięto dodatkowe możliwości konfiguracji układu programowalnego (*Bypass*). Zatwierdzono układ XC3S500E oraz naciśnięto opcję *Program*. Po poprawnym zaprogramowaniu układu FPGA ukazała się informacja *Program Succeeded*.

### 5.6.2. Konfiguracja z wykorzystaniem układu Xilinx XCF04S i pliku Flash PROM

Analogicznie jak w poprzedniej konfiguracji na początku wygenerowano plik .bit używając polecenia *Generate Programming File*. Po lewej stronie okna środowiska wybrano *Create PROM File (PROM File Formatter)*. Ukazało się kolejne okno, w którym dokonano następującej konfiguracji:

* Typ urządzenia: Xilinx Flash/PROM,
* Rodzina pliku PROM (ang. PROM Family): Platform Flash,
* Urządzenie (ang. Device): xcf04s (4M),
* Nazwa pliku (ang. Output file name): slave\_fifo\_prom,
* Lokalizacja pliku (ang. Output file location): standardowy katalog projektu,
* Format pliku (ang. File format): MCS.

Po zaakceptowaniu ustawień przypisano plik .bit do urządzenia oraz wygenerowano plik PROM wybierając opcję *Operations - Generate File*. Po prawej stronie okna ukazało się potwierdzenie *Generate Succeeded* (Rys. 27).



Rys. . Wygenerowanie pliku PROM

Po wygenerowaniu pliku PROM wybrano opcję *Boundary Scan* oraz zainicjalizowano łańcuch dostępnych układów identycznie jak w poprzednim podrozdziale. Tym razem do układu docelowego nie przypisywano pliku .bit, ale do XCF04S przypisano wygenerowany plik .mcs. Po wybraniu go i zaprogramowaniu otrzymano komunikat *Program Succeeded*.

# 6. Prezentacja wyników oraz efektów działania aplikacji

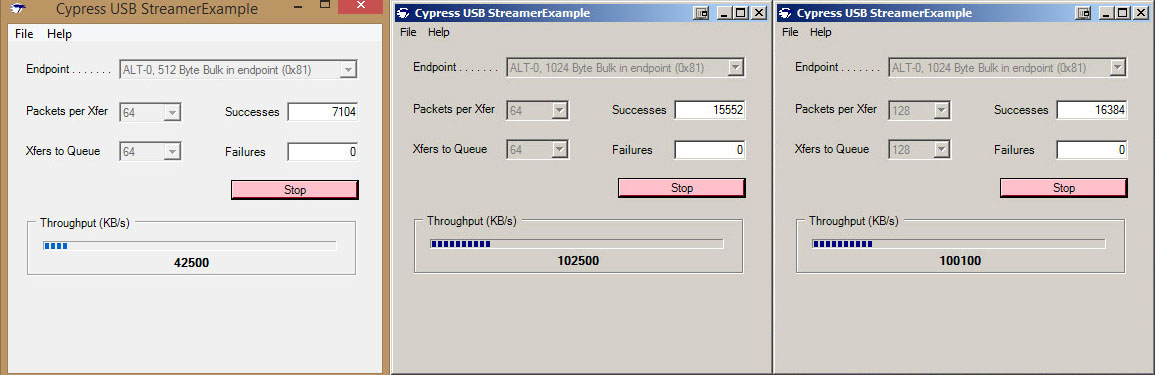
W poprzednich rozdziałach zaprezentowano teorię oraz praktyczną realizację oprogramowania dla obu zestawów ewaluacyjnych. Poniżej przedstawiono efekty działania aplikacji.

## 6.1. Transmisja strumienia danych z układu FPGA do komputera

Po uruchomieniu i zaprogramowaniu obu układów użyto przełącznika oznaczonego na zestawie uruchomieniowym Spartan 3E kodem SW2, dzięki któremu uruchomiono tryb strumieniowego transferu danych z FPGA do komputera. Na wyświetlaczu LCD zaobserwowano napis:

FSM: STREAM IN  
0000111100001111

W drugiej linii LCD zdecydowano wyświetlać dwa bajty danych, które program w danym momencie wysyła do komputera. Dopóki nie skonfigurowano programu Cypress Streamer odpowiadającego za strumieniowy transfer danych, program znajdował się w stanie bezczynności. Z tego powodu zaobserwowano dwa bajty 0x0F0F, które mają za zadanie sygnalizować brak transferu danych do komputera.



Rys. . Strumieniowy transfer danych z wykorzystaniem programu Cypress Streamer

Uruchomiono dedykowany program dostarczony przez firmę Cypress, a następnie dokonano konfiguracji ustawień. Wybrano wejściowy punk końcowy (*Bulk in endpoint 0x81*), liczbę pakietów oraz transferów ustawiono odpowiednio na 64 i 128 (Rys. 28). Naciśnięto przycisk *Start* i zauważono zmianę napisu na wyświetlaczu LCD:

FSM: STREAM IN  
0101011101001101

Uznano zmianę dwóch bajtów z 0x0F0F na 0x574D jako potwierdzenie, że dane z układu programowalnego były wysyłane prawidłowo. Dokonano weryfikacji poprawnego odbioru danych w komputerze na kilka sposobów.

### 6.1.1. Test prędkości transferu danych

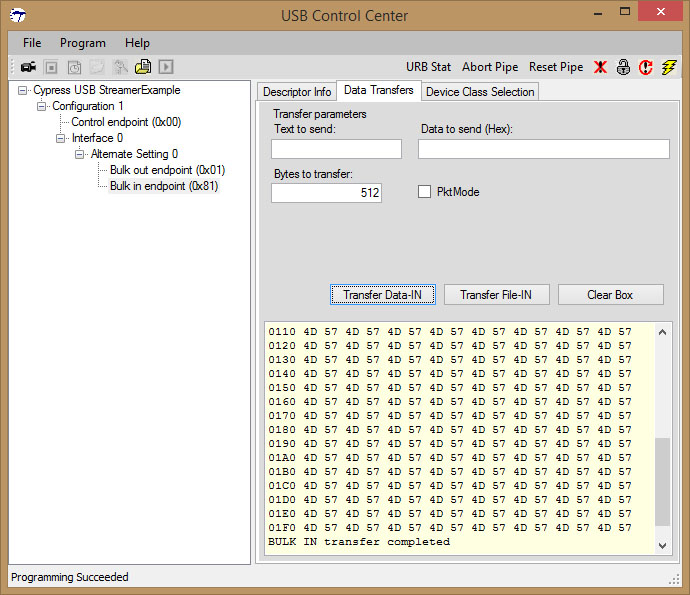
Na Rys. 28 przedstawiono okno programu Cypress Streamer podczas strumieniowego transferu danych. Dla okna znajdującego się po lewej stronie oraz w środku zastosowano identyczną liczbę pakietów oraz transferów (64), jednakże wykorzystano inny standard magistrali USB, odpowiednio USB 2.0 oraz USB 3.0. Zauważono znaczną różnicę w transferze, dla USB 2.0 transfer odbywał się z prędkością 42500 kB/s, natomiast przy wykorzystaniu USB 3.0 prędkość zwiększyła się do 102500 kB/s.

Po prawej stronie Rys. 28 umieszczono okno z podwojoną liczbą wysyłanych pakietów oraz transferów (opcja *Packets per Xfer* oraz *Xfers to Queue*) w stosunku do środkowego okna. W tym przypadku wykorzystano również standard USB 3.0, jednak zwiększenie ustawień z 64 na 128 nie zwiększyło prędkości transferu, co zgadza się z Wyk. 1.

Dla trzech przeprowadzonych transferów danych nie zanotowano błędnego odbioru danych w komputerze (okno *Failures*), natomiast liczba poprawnie odebranych pakietów rosła wraz z upływającym czasem (okno *Successes*).

### 6.1.2. Odczytanie pakietu danych w programie USB Control Center

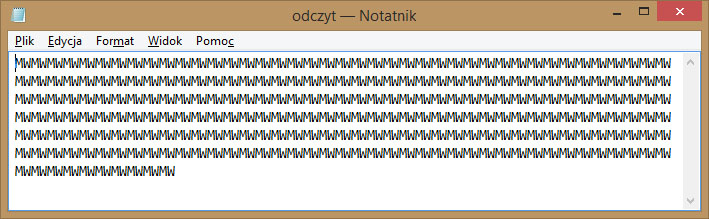
Wybrano wejściowy punkt końcowy z drzewa konfiguracji (*Bulk in 0x81*) programu USB Control Center, aby móc bezpośrednio odczytać pakiet danych. Następnie przełączono się na zakładkę *Data Transfers* i wybrano przycisk *Transfer Data-IN*. W oknie głównym zaobserwowano pakiet danych zapisany w systemie heksadecymalnym (Rys. 29). Składa się dwóch bajtów zdefiniowanych w FPGA (0x574D).



Rys. . Pobranie pakietu danych z FPGA

### 6.1.3. Zapianie pakietu danych do pliku

Zapisano odebrane dane do pliku tekstowego wybierając opcję *Transfer File-IN* w programie USB Control Center. Następnie plik otworzono w domyślnym edytorze tekstowym systemu Windows (Rys. 30).



Rys. . Zapisany pakiet danych do pliku, dwa bajty 0x574D odpowiadają literom MW w kodzie ASCII

## 6.2. Odbiór strumienia danych przez układ FPGA

Przywrócono przełącznik SW2 do domyślnej pozycji, wybrano przełącznik oznaczony jako SW1, aby program FPGA uruchomił tryb odbioru strumienia danych z komputera. W programie Cypress Streamer wybrano wyjściowy punkt końcowy (*Bulk out 0x01*), liczba pakietów oraz transferów pozostała niezmieniona (64). Zanotowano na wyświetlaczu LCD napis:

FSM: STREAM OUT  
1111000011110000

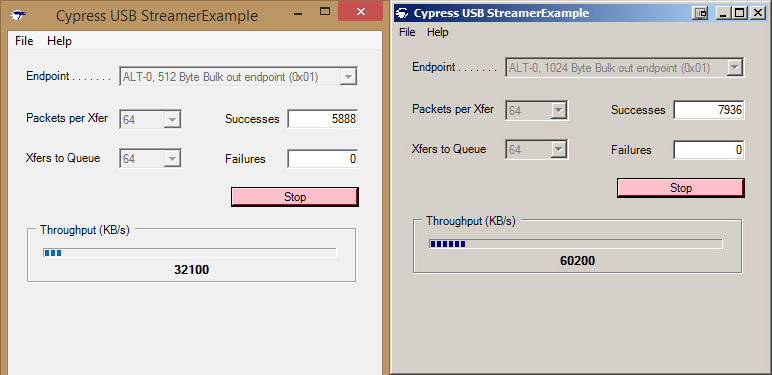
Zastosowano ciąg 0xF0F0 zamiast 0x0F0F, aby odróżnić stan bezczynności transmisji nadawania i odbioru danych. Naciśnięto przycisk *Start* w programie Cypress Streamer (Rys. 31) i zaobserwowano zmianę napisu w drugiej linii na wyświetlaczu LCD:

FSM: STREAM OUT  
0000000000000000

Zmiana dwóch bajtów z 0xF0F0 na 0x0000 była potwierdzeniem, że układ programowalny rozpoczął transmisję i odbierał dane bezproblemowo.

### 6.2.1. Test prędkości transferu danych

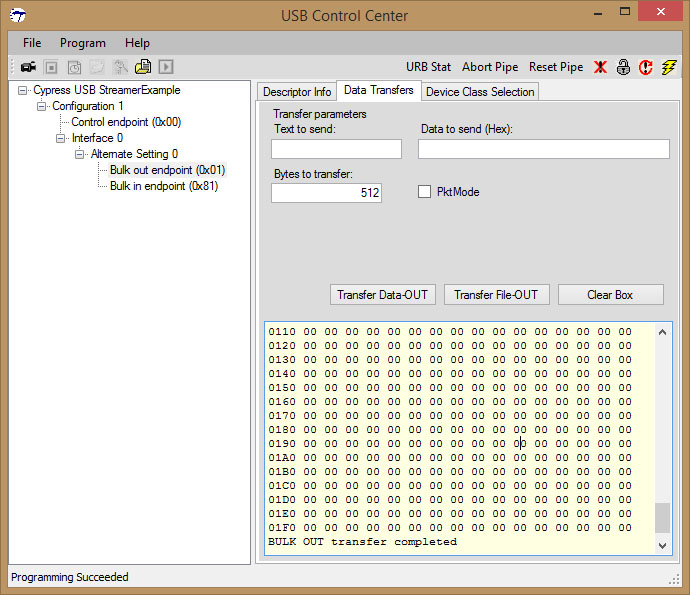
Na Rys. 31 zamieszczono okna programu Cypress Streamer, po lewej z wykorzystaniem standardu USB 2.0, a po prawej USB 3.0. Prędkość transferu strumieniowego wysyłania danych z komputera do FPGA wynosiła 32100 kB/s dla USB 2.0. Podczas wykorzystania USB 3.0 zanotowano prawie dwukrotnie większą prędkość, czyli 60200 kB/s.



Rys. . Strumieniowa transmisja danych z komputera do FPGA

### 6.2.2. Wysłanie pakietu danych w programie USB Control Center

Wykorzystując narzędzie USB Control Center możliwy był również transfer pakietu do FPGA. Po wyborze odpowiedniego punktu końcowego (*Bulk out, 0x01*) i naciśnięciu przycisku *Transfer Data-OUT* zanotowano w głównym oknie ciąg danych złożony z samych zer (Rys. 32).



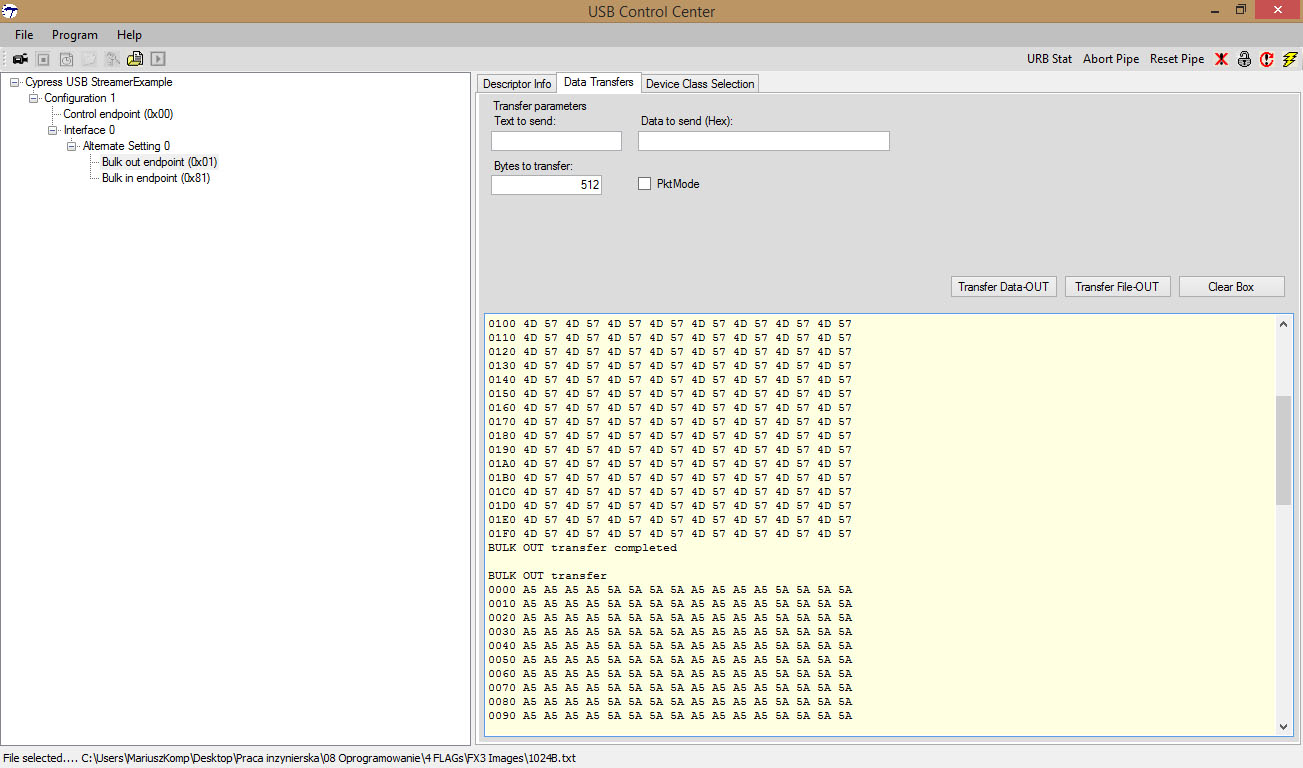
Rys. . Transmisja pakietu danych do FPGA

## 6.3. Transmisja danych w obu kierunkach

Jest to najbardziej złożony tryb transmisji danych polegający na wysłaniu pakietu (standardowo 512 bajtów) danych z komputera do FPGA, zapisanie ich w pamięci FIFO oraz odesłanie z powrotem do komputera. Aby uruchomić tryb transferu danych w obu kierunkach należało układ wyłączyć oraz włączyć ponownie w celu wykasowania programu w pamięci RAM (lub nacisnąć przycisk *Reset* oznaczony jako SW8 na płytce uruchomieniowej). Po załadowaniu odpowiedniego pliku .img do pamięci RAM przystąpiono do zbadania poprawności działania transferu.

Na zestawie uruchomieniowym Spartan 3E przywrócono przełącznik SW1 do domyślnej pozycji i następnie załączono przełącznik oznaczony jako SW0. Odpowiedzialny jest on za wejście do trybu transferu danych w obu kierunkach ze stanu bezczynności.

Wysłanie i późniejszy odbiór danych umożliwia jedynie narzędzie USB Control Center. Przygotowano dwa pliki z danymi, pierwszy o rozmiarze 512 bajtów, a drugi dwa razy większy, czyli o rozmiarze 1024 bajtów. Po zaznaczeniu wyjściowego punktu końcowego (*Bulk out, 0x81*) i wybraniu opcji *Transfer File-OUT* wysłano do FPGA trzy pakiety danych po 512 bajtów. W pierwszej kolejności wysłano pakiet 0x574D, a potem dwa pakiety 0xA5A5 (Rys. 33).



Rys. . Dokonano transferu 3 pakietów danych o rozmiarze 512 bajtów

Po wysłaniu 1536 bajtów danych wyczyszczono okno wybierając opcję *Clear Box*. Następnie wpisano w okno *Bytes to transfer* liczbę 1536 i naciśnięto przycisk *Transfer Data-IN*. Na początku zarejestrowano ciąg złożony z bajtów 0x574D, a w dalszej kolejności zanotowano dane składające się z powtarzającej się sekwencji 0xA5A5. Odczytana kolejność odebranych danych zgadza się z zasadą działania kolejki FIFO (ang. First In, First Out, czyli pierwsze na wejściu, pierwsze na wyjściu).

## 6.4. Porównanie szybkości transferu danych w zależności od ilości przesyłanych pakietów

W programie Cypress Streamer użytkownik posiada możliwość manipulacji opcjami, które odpowiadają za prędkość transferu [8].

* **Packets per Xfer** - liczba pakietów przeznaczona do transferu danych. Im większa liczba pakietów w pojedynczym transferze, tym uzyskuje się większą przepustowość interfejsu,
* **Xfers to Queue** - zwiększa liczbę transferów oraz dodaje je do kolejki zadań. Opcja redukuje opóźnienie występujące pomiędzy udanymi transferami.

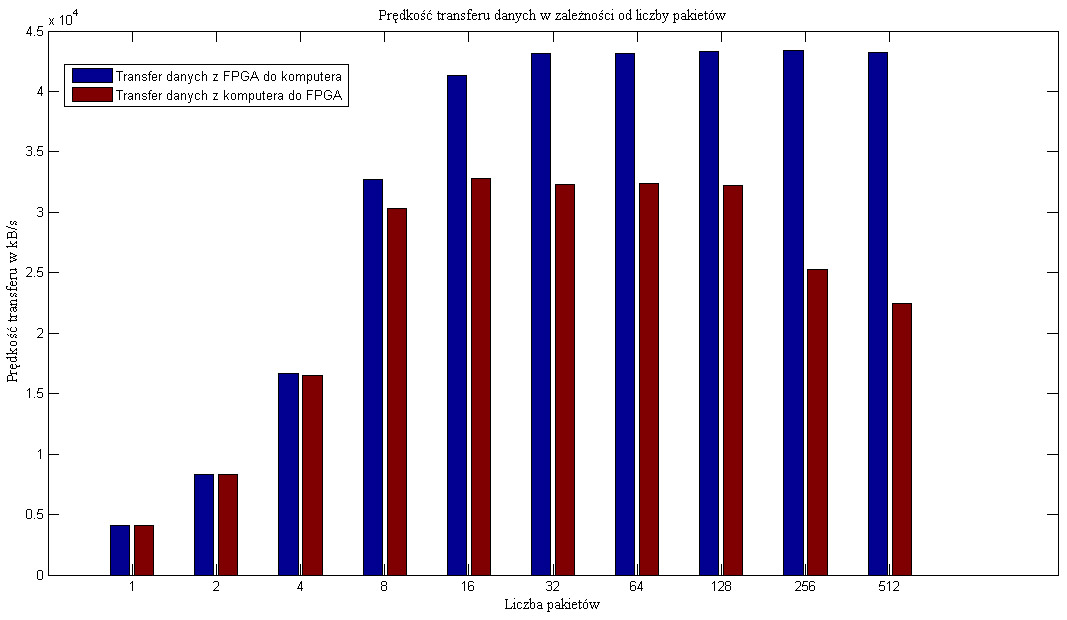
Manipulowano liczbą pakietów (od 1 do 512) przesyłanych podczas transferu (*Packets per Xfer*). Dla opcji *Xfers to Queue* ustawiono stałą wartość, która wynosiła 128. Wyniki przedstawiono w Tab. 12 oraz na Wyk. 1.

Tab. . Badanie zależności pomiędzy prędkością transferu a liczbą przesyłanych pakietów

|  |  |  |  |
| --- | --- | --- | --- |
| **Liczba pakietów** | **Liczba transferów w kolejce** | **Transfer danych z FPGA do komputera (kB/S)** | **Transfer danych z komputera do FPGA (kB/s)** |
| 1 | 128 | 4100 | 4100 |
| 2 | 8300 | 8300 |
| 4 | 16700 | 16500 |
| 8 | 32700 | 30300 |
| 16 | 41300 | 32800 |
| 32 | 43100 | 32300 |
| 64 | 43100 | 32400 |
| 128 | 43300 | 32200 |
| 256 | 43400 | 25300 |
| 512 | 43200 | 22500 |

Dla strumieniowego transferu danych z FPGA do komputera zauważono, że wraz ze zwiększaniem liczby pakietów rośnie również przepustowość interfejsu. Powyżej 8 pakietów prędkość transferu stabilizuje się i wynosi około 43100 kB/s.

Odmienne wnioski wyciągnięto dla drugiej transmisji, która opierała się na odbieraniu pakietów danych w układzie programowalnych wysyłanych strumieniowo z komputera. Również zanotowano wzrost prędkości transferu dla początkowych wartości pakietów, jednakże po ustawieniu 256 pakietów zauważono radykalny spadek prędkości transferu. Po zwiększeniu liczby pakietów do wartości maksymalnej (512) transfer wynosił 22500 kB/s. Prędkość maksymalna dla 64 pakietów była zdecydowanie niższa od uzyskanej prędkości transferu danych do komputera z FPGA.



Wyk. . Prędkość transferu w zależności od liczby pakietów

# 7. Podsumowanie

W ramach pracy zrealizowano interfejs synchronicznej kolejki FIFO z wykorzystaniem programowalnej maszyny stanów GPIF II, która dzięki współpracy z magistralą USB 3.0 daje możliwość szybkiej wymiany danych pomiędzy układem FPGA a komputerem stacjonarnym. Wykorzystano zestaw uruchomieniowy firmy Xilinx Spartan 3E oraz zestaw Cypress CYUSBKIT-001. Synchroniczny interfejs posiada 16-bitową szyną danych, jednakże może być ona zwiększona do 32-bitów jeśli zastosuje się FPGA z większą liczbą dostępnych wyprowadzeń.

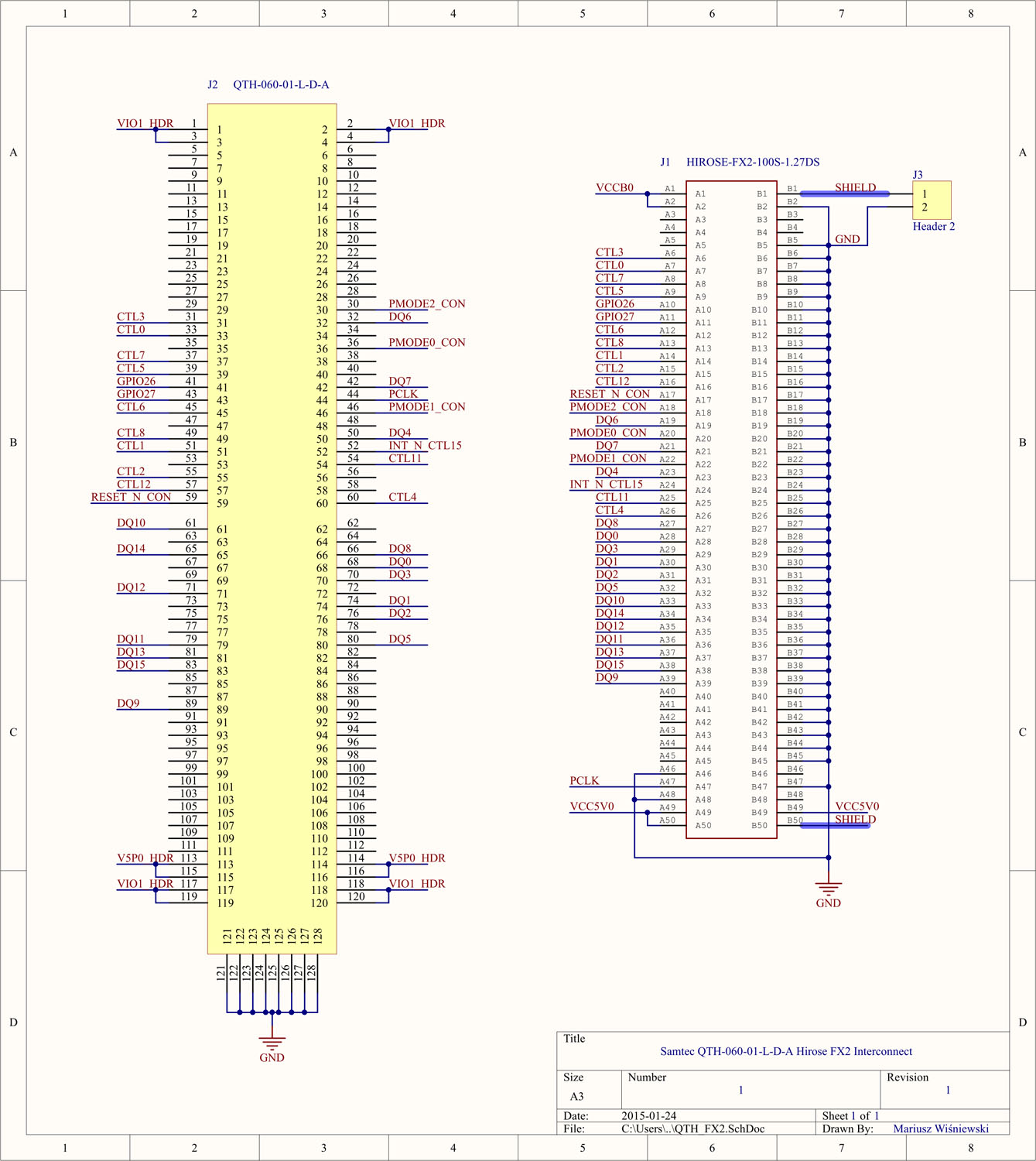
Po zrealizowaniu pracy stwierdzono, że główny cel został osiągnięty. Dzięki dodaniu interfejsu synchronicznej kolejki FIFO do układu programowalnego istnieje możliwość transferu danych (w obu kierunkach) z wykorzystaniem magistrali USB 3.0 oraz USB 2.0. Transfer strumieniowy z FPGA do komputera przy wykorzystaniu USB 2.0 osiągnął prędkość 42,5 MB/s, a z wykorzystaniem USB 3.0 prędkość transferu wyniosła maksymalnie 100 MB/s. Z kolei transfer danych z komputera do FPGA osiągnął 32 MB/s przy wykorzystaniu USB 2.0, a w standardzie 3.0 zanotowano prędkość wynoszącą 60 MB/s.

# 8. Literatura

1. Universal Serial Bus Common Class Specification, Rev. 1.0, 1997
2. T. Francuz, AVR Praktyczne Projekty, Helion, wydanie elektroniczne 2013
3. J. Axelson, *USB Complete Fourth Edition: The Developer's Guide*, Lakeview Research, Madison 2009
4. FX3 Programmers Manual, Doc. 001-64707, Rev. G, 2013
5. EZ-USB® FX3/FX3S SDK Firmware API Guide, Version 1.3.1, 2013
6. Getting Started with EZ-USB® FX3™, Doc. 001-75705, Rev. C, 2013

1. <http://www.cypress.com/?rID=58321> - strona firmy Cypress opisująca CYUSB3KIT-001
2. CYUSB3KIT-001 EZ-USB®FX3™ Development Kit Guide, Doc. 001-70237, Rev. C, 2014
3. Designing With the EZ-USB® FX3™ Slave FIFO Interface, Doc. 001-65974, Rev. J, 2014
4. Designing a GPIF™ II Master Interface, Doc. 001-87216, Rev. A, 2013
5. J. Majewski, P. Zbysiński, *Układy FPGA w przykładach*, BTC, Warszawa 2007
6. Spartan-3E FPGA Starter Kit Board User Guide, UG230 (v1.2), 2011
7. Using Digital Clock Managers (DCMs) in Spartan-3 FPGAs, XAPP462 (v1.1), 2006
8. LogiCORE IP FIFO Generator v9.1, DS317, 2012
9. Spartan-3E FPGA Family, DS312, 2013
10. Xilinx Spartan 3E Starter Board Schematics, Doc. 500-087, Rev. D, 2006
11. CYUSB3ACC-002 FMC Interconnect Board Schematics, Doc. 630-60187-01, Rev. 1, 2014
12. SOP Connector Rework Methods, Doc. 1-800-SAMTEC-9, Rev. 2/2/2005

# 9. Załącznik



Zał. . Schemat elektryczny obwodu drukowanego łączącego dwa zestawy uruchomieniowe