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  
Dd

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

Spis treści

[1. Wstęp 6](#_Toc410728319)

[1.1. Cel pracy 6](#_Toc410728320)

[1.2. Lista używanego oprogramowania 7](#_Toc410728321)

[2. Wprowadzenie do USB 8](#_Toc410728322)

[2.1. Standard USB 2.0 8](#_Toc410728323)

[3. Interfejs synchronicznej kolejki FIFO oraz programowalna maszyna stanów GPIF II 9](#_Toc410728324)

[3.1. Interfejs synchronicznej kolejki FIFO 9](#_Toc410728325)

[3.2. Płytka łącząca dwa zestawy uruchomieniowe 10](#_Toc410728326)

[3.3. Programowalna maszyna stanów GPIF II 14](#_Toc410728327)

[3.4. Narzędzie GPIF II Designer 14](#_Toc410728328)

[3.4.1. Utworzenie i edycja sygnałów interfejsu 14](#_Toc410728329)

[3.4.2. Edycja maszyny stanów 16](#_Toc410728330)

[3.4.3. Edycja flag pełnych oraz częściowych 16](#_Toc410728331)

[3.4.4. Kompilacja projektu 17](#_Toc410728332)

[4. Cypress EZ-USB FX3 18](#_Toc410728333)

[4.1. Teoria FX3 - wątki, gniazda, DMA oraz deskryptory 20](#_Toc410728334)

[4.1.1. Gniazda 20](#_Toc410728335)

[4.1.2. Deskryptory DMA 20](#_Toc410728336)

[4.1.3. Bufor DMA 20](#_Toc410728337)

[4.1.4. Wątki GPIF II 21](#_Toc410728338)

[4.1.5. Konfiguracja kanału DMA 21](#_Toc410728339)

[4.2. Konfiguracja flag pełnych oraz częściowych 22](#_Toc410728340)

[4.2.1. Flagi dedykowane, związane z konkretnym wątkiem 22](#_Toc410728341)

[4.2.2. Flagi wskazujące na aktualny wątek 23](#_Toc410728342)

[4.2.3. Flaga częściowa 23](#_Toc410728343)

[4.3. Interfejs synchronicznej kolejki FIFO - realizacja oprogramowania dla FX3 23](#_Toc410728344)

[4.3.1. Zintegrowane środowisko programistyczne - EZ-USB Suite 24](#_Toc410728345)

[4.3.2. Ciągłe wysyłanie lub odbieranie danych 24](#_Toc410728346)

[4.3.3. Pętla 26](#_Toc410728347)

[4.4. USB Control Center - programowanie układu 26](#_Toc410728348)

[5. Xilinx Spartan 3E 28](#_Toc410728349)

[5.1. Środowisko projektowe 28](#_Toc410728350)

[5.2. Utworzenie projektu, przypisanie pinów 28](#_Toc410728351)

[5.3. Cyfrowe Zarządzanie Zegarem (DCM - Digital Clock Manager) 30](#_Toc410728352)

[5.4. Pamięć FIFO 33](#_Toc410728353)

[5.5. Ogólna struktura programu 34](#_Toc410728354)

[5.5.1. Główny program - Slave Fifo Main 34](#_Toc410728355)

[5.5.2. Ciągłe wysyłanie danych do komputera - Stream Write to FX3 36](#_Toc410728356)

[5.5.3. Ciągłe odbieranie danych do komputera - Stream Read from FX3 37](#_Toc410728357)

[5.5.4. Pętla - Loopback Transfer 38](#_Toc410728358)

[5.6. Programowanie FPGA 39](#_Toc410728359)

[5.6.1. Konfiguracja z wykorzystaniem interfejsu JTAG 39](#_Toc410728360)

[5.6.2. Konfiguracja z wykorzystaniem układu Xilinx XCF04S i pliku Flash PROM 41](#_Toc410728361)

[6. Przedstawienie wyników oraz efektów działania interfejsu 43](#_Toc410728362)

[6.1. Ciągłe wysyłanie danych z FPGA do komputera 43](#_Toc410728363)

[6.2. Ciągłe wysyłanie danych z PC do FPGA 45](#_Toc410728364)

[6.3. Pętla 46](#_Toc410728365)

[8. Podsumowanie 48](#_Toc410728366)

# 1. Wstęp

Uniwersalną magistralę szeregową (USB) opracowano w 1996 roku, a rok później zaczęto ją wdrażać do komputerów stacjonarnych na masową skalę. W chwili obecnej każdy komputer obsługuje USB 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. 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, systemy wbudowane wykorzystujące układy oparte na rdzeniu ARM (komercyjny sukces odniosły płytki Raspberry Pi, Cubieboard) oraz przede wszystkim firmy produkujące inteligentne telefony komórkowe. Nie trzeba już stosować dodatkowych programatorów lub konwerterów USB w celu zaprogramowania procesora. Zmniejsza to poniesione koszty dla użytkowników końcowych. USB wypiera zarówno w profesjonalnym jak i w amatorskim zastosowaniu magistrale o słabej przepustowości, na przykład popularny RS-232, I2C czy SPI.

## 1.1. Cel pracy

Niniejsza praca ma na celu dodanie możliwości transferu danych wykorzystując magistralę USB (w celu ich gromadzenia lub wizualizacji) do układu, który jej standardowo nie obsługuje. Wykorzystano do tego celu komputer, kontroler EZ-USB FX3 firmy Cypress i zestaw ewaluacyjny oparty na układzie programowalnym FPGA Spartan 3E firmy Xilinx. Płytkę łączącą oba zestawy wykonano samodzielnie.

Rozdział drugi ma na celu przybliżyć standard USB, począwszy od USB 1.0 aż do najnowszej wersji, czyli 3.0. Omówiono w nim topologię oraz najistotniejsze elementy składające się na magistralę. Następnie wyjaśniono poszczególne rodzaje transmisji danych, takie jak kontrolna, przerwaniowa, masowa lub izochroniczna.

Rozdział trzeci skupia się na omówieniu interfejsu synchronicznej kolejki FIFO (ang. Synchronous Slave FIFO Interface) oraz programowalnej maszyny stanów GPIF II. Zaprezentowano najważniejsze sygnały oraz koncept użycia ich podczas transmisji. Na potrzeby pracy dyplomowej zaprojektowano płytkę łączącą dwa układy. Schematy, fotografie oraz projekt obwodu drukowanego (PCB) również zamieszono w tym dziale. Pod koniec opisano 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 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, jeden dotyczył ciągłego wysyłania lub odbierania danych, a drugi pętli. Dodatkowo opisano jak korzystać z narzędzia USB Control Center służącego do programowania układu.

Rozdział piąty opisuje program napisany dla FPGA Spartan 3E. Kod oparty został na maszynach o skończonej liczbie stanów, dzięki którym przebiega transmisja danych. Dla każdej transmisji przesyłania danych zostały przedstawione diagramy i opisy zmiany stanów sygnałów zarządzających magistralą. Został również omówiony użyty ekran LCD, który sygnalizuje obecny stan oraz wysyłane lub odbierane bajty danych. Ponadto skupiono się na zaimplementowanych modułach przygotowanych przez firmę Xilinx, takich jak Cyfrowe Zarządzanie Zegarem oraz pamięć FIFO. Na koniec przedstawiono dwie możliwości programowania układu.

W ostatnim rozdziale zamieszczono efekty transmisji danych, czyli zrzuty ekranów programów USB Control Center oraz Cypress Streamer.

## 1.2. Lista używanego oprogramowania

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

Tab. . Lista użytego oprogramowania wraz z opisem

|  |  |
| --- | --- |
| **Nazwa** | **Opis** |
| Altium Designer 14.3.9 | projekt płytki PCB łączącej FPGA z FX3 |
| Webpack ISE 14.1 | synteza, kompilacja projektu dla FGPA |
| PlanAhead | przypisanie pinów do układu FPGA |
| ISE Impact | programowanie układu programowalnegowykorzystując interfejs JTAG |
| GPIF II Designer | zdefiniowanie ustawień oraz maszyny stanów dla synchronicznej kolejki FIFO |
| Cypress EZ-USB Suite | kompilacja i edycja projektu dla EZ-USB FX3 |
| USB Control Center | programowanie FX3, odbiór i wysyłanie danych |
| Cypress Streamer | masowy transfer danych wykorzystujący USB |
| Git 1.9.4 | rozproszony system kontroli wersji |

# 2. Wprowadzenie do USB

USB jest rozwiązaniem idealnym jeśli istnieje potrzeba, aby komputer komunikował się z zewnętrznymi urządzeniami. Interfejs sprawdza się zarówno w projektach produkowanych na skalę przemysłową jak i w amatorskich konstrukcjach. Poniżej przedstawiono korzyści jakie płyną z zastosowania tej magistrali.

Korzyści dla użytkowników końcowych:

* łatwość obsługi - istnieje jeden standard dla wielu urządzeń, wytrzymałe konektory, automatyczna konfiguracja urządzenia w systemie, możliwość działania bez względu na system operacyjny, możliwość rozszerzenia portów USB przy użyciu HUB-ów, prostota instalacji sterowników jeśli zajdzie taka potrzeba
* 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. 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 dołączanych urządzeń, inicjalizuje i zarządza transferem danych. Można rozszerzyć tę liczbę 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).

Wspierane są trzy prędkości przesyłu:

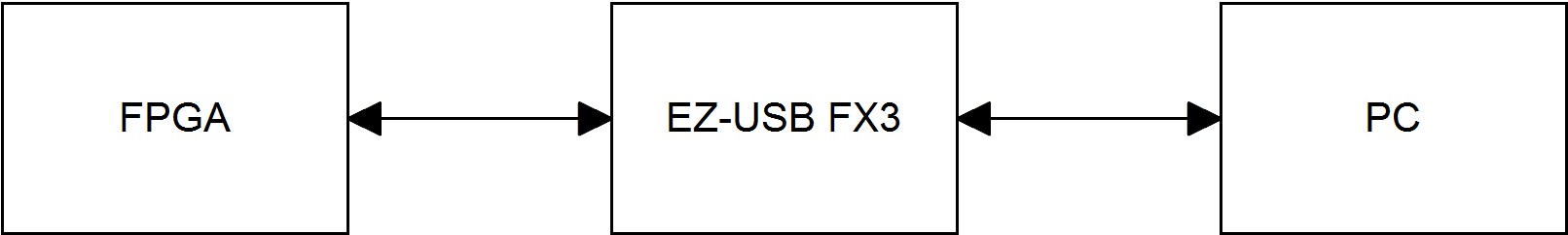
* low speed - transfer z maks. prędkością 1,5 Mbit/s, opracowany w standardzie USB 1.0
* full speed - transfer z maks. prędkością 12 Mbit/s, opracowany w standardzie USB 1.1
* high speed - transfer z maks. prędkością 480 Mbit/s, opracowany w 2001 roku (USB 2.0)

Poniżej zostaną przedstawione najważniejsze

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

Jednym z najpopularniejszych interfejsów, który można wykorzystać w urządzeniu EZ-USB FX3 firmy Cypress jest implementacja synchronicznej kolejki FIFO. Ten rodzaj transmisji posiada bardzo bogatą dokumentację dołączoną do zestawu CYUSBKIT-001 na płycie CD oraz na oficjalnej stronie producenta. 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 (First In, First Out, czyli pierwszy na wejściu, pierwszy na wyjściu).

Poniższy schemat (Rys. 1) przedstawia schemat działania docelowej aplikacji:



Rys. . Schemat działania aplikacji

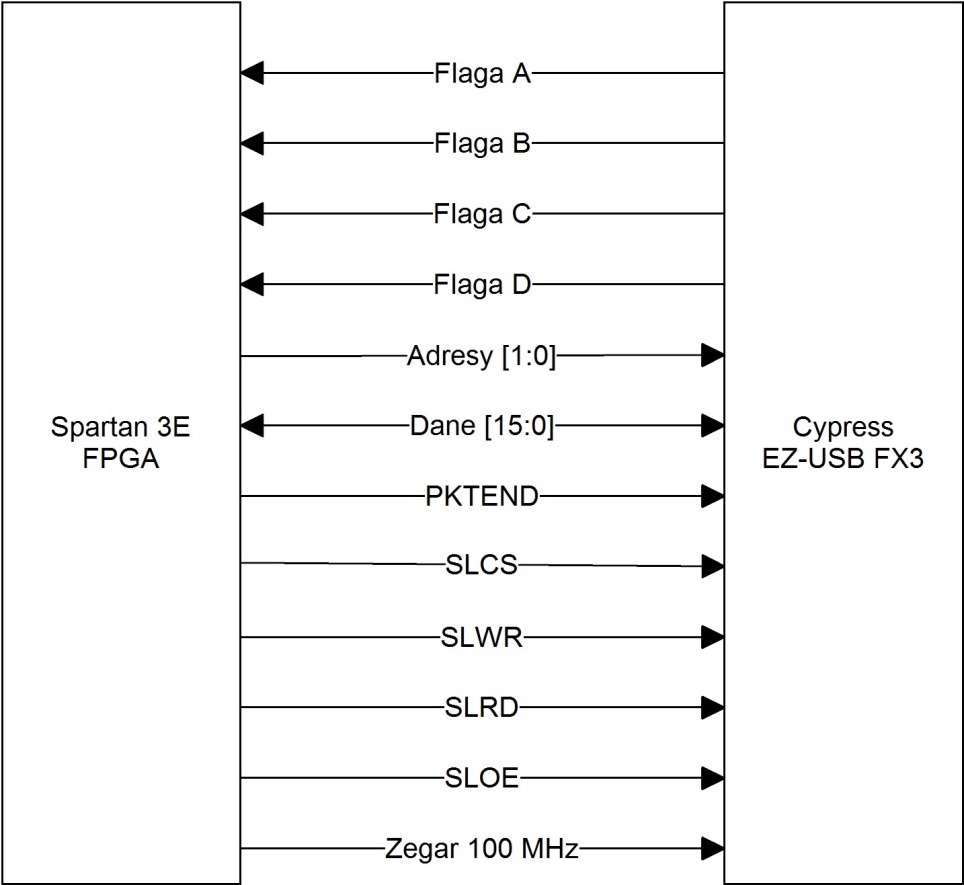
## 3.1. Interfejs synchronicznej kolejki FIFO

Wybór tego interfejsu jest idealnym rozwiązaniem, gdy zewnętrze urządzenie (dowolny procesor lub układ programowalny) standardowo nie posiada magistrali USB, a zachodzi potrzeba komunikacji z komputerem. Można w tym przypadku zastosować komunikację RS-232, ale użytkownik jest ograniczony słabą przepustowością magistrali. Urządzenie EZ-USB FX3 rozwiązuje ten problem dodając do aplikacji magistralę USB.

Diagram (Rys. 2) prezentuje schematycznie użyte sygnały, a poniżej (Tab. 2) znajduje się ich szczegółowy opis.

Tab. . Sygnały synchronicznej kolejki FIFO

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



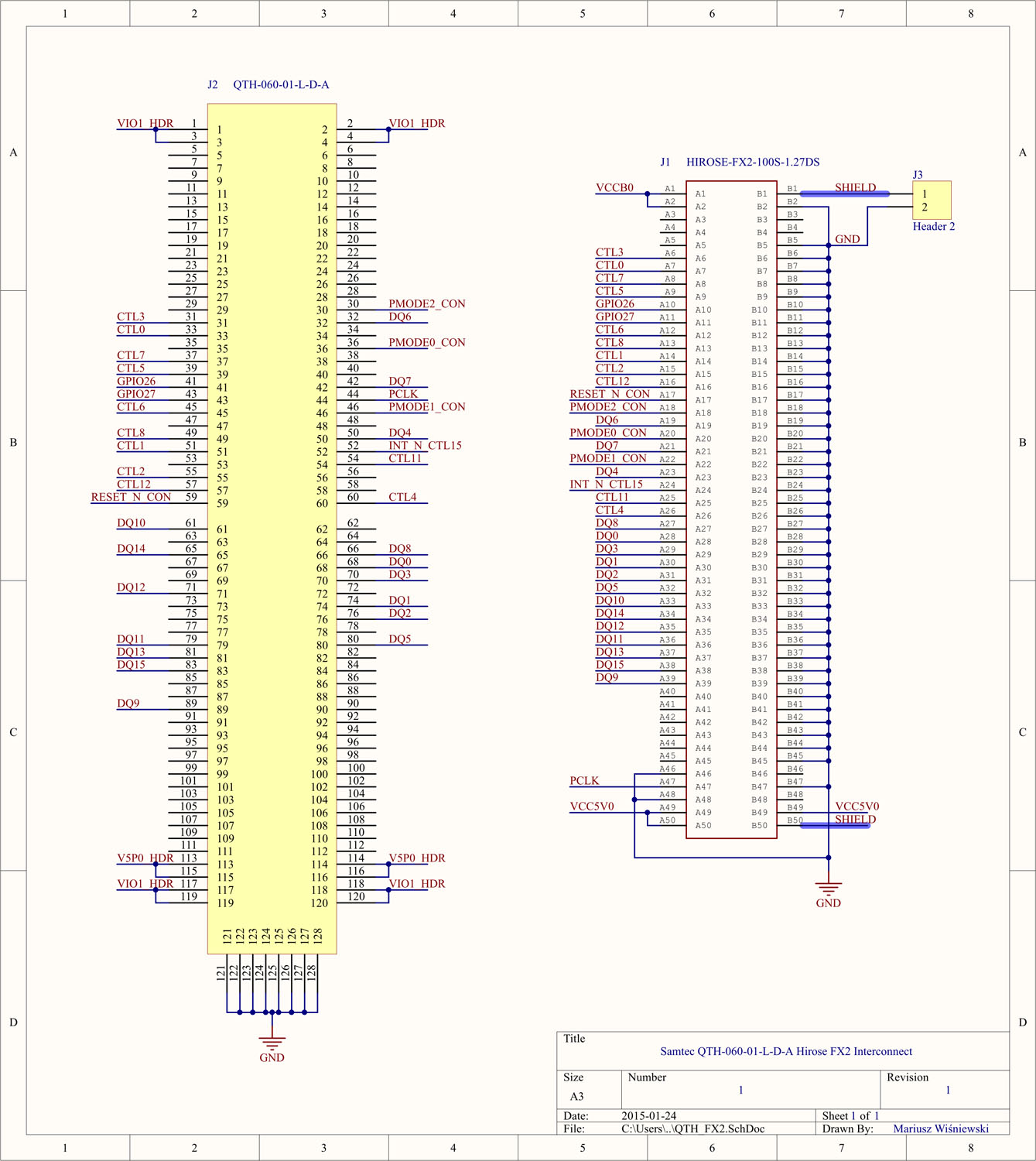
Rys. . Schemat działania interfejsu synchronicznej kolejki FIFO

Interfejs synchronicznej kolejki FIFO 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 następujące konsekwencje, m.in.: dodatkowe opóźnienia podczas monitorowania stanu flagi oraz po zmianie adresu. W omawianej pracy ten przypadek nie jest rozpatrywany.

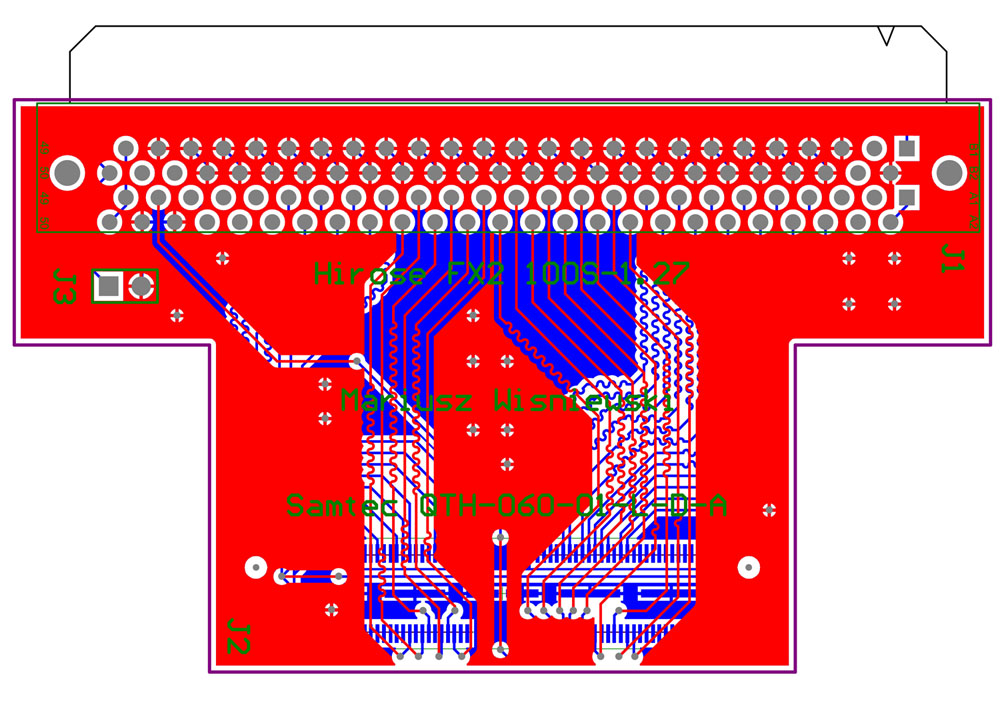
## 3.2. Płytka łącząca dwa zestawy uruchomieniowe

Firma Cypress nie posiada w swojej ofercie płytki, która pozwoli na sprzęgnięcie ze sobą obu używanych zestawów w pracy. W sklepie internetowym producenta FX3 można odnaleźć jedynie płytkę o nazwie CYUSB3ACC-002 (zaprzestano jej produkcji, niedostępna w sprzedaży). Ponadto przeznaczona jest jedynie dla układów posiadających złącze FMC (Xilinx LPC).

Schemat elektryczny oryginalnej płytki oraz dokumentacja elektryczna zestawu Spartan 3E posłużyły jako baza do zaprojektowania własnej przejściówki, opartej na złączach Hirose FX2 oraz Samtec QTH-060-01-L-D-A. Schemat ideowy (Rys. 3) oraz projekt PCB (Rys. 4) został wykonany w programie Altium Designer, ze względu na możliwości, które oferuje to oprogramowanie. Wszystkie ścieżki (oprócz linii zegarowej) posiadają identyczną długość. Ma to na celu zlikwidowanie ewentualnych opóźnień na magistrali danych i eliminację efektu propagacji sygnałów. Zaprojektowana płytka jest dwuwarstwowa.

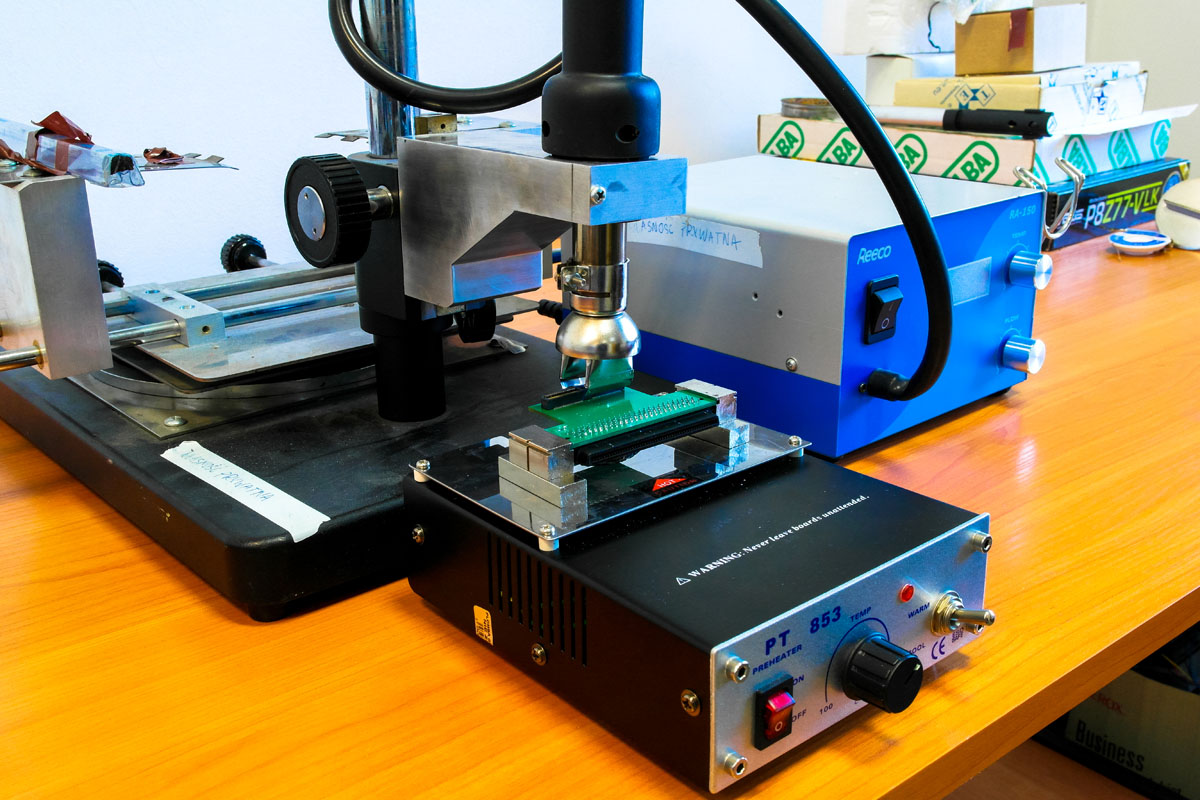


Rys. . Schemat elektryczny płytki



Rys. . Projekt PCB płytki

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.



Rys. . Proces montażu elementów gorącym powietrzem. Widoczny podgrzewacz oraz lutownica hot-air

Do lutowania metodą na gorące powietrze posłużyły następujące narzędzia (Rys. 5):

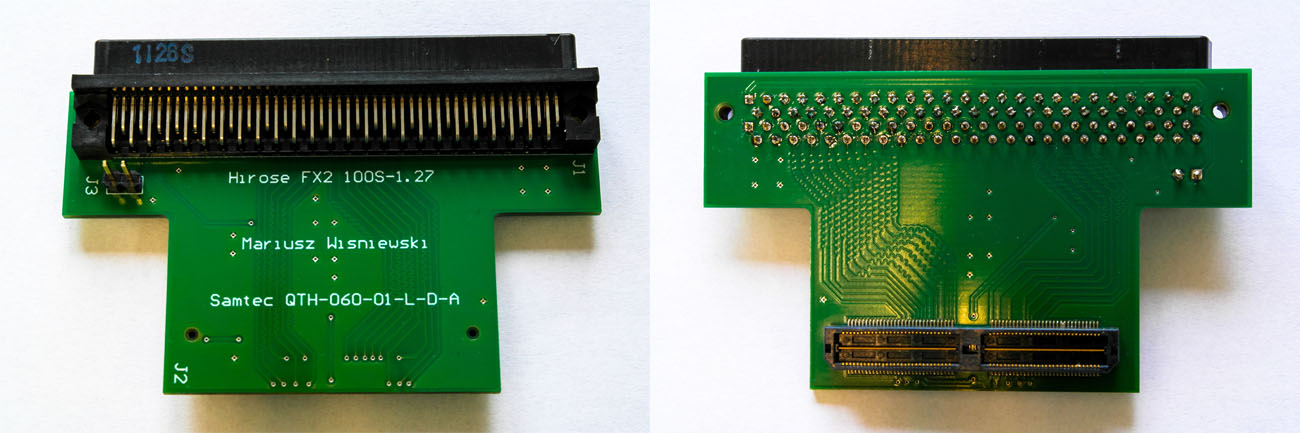
* podgrzewacz PT 853
* stacja hot-air Reeco RA-150

Poniżej przedstawiono fotografię efektu końcowego, czyli gotową płytkę sprzęgającą dwa układy (Rys. 6).



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

Zdjęcia samej płytki z obu stron (Rys. 7):



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

## 3.3. Programowalna maszyna stanów GPIF II

GPIF II to programowalna maszyna stanów, która wprowadza możliwość implementacji dowolnego standardu komunikacji (własnego lub wcześniej opracowanego przez korporacje). Może funkcjonować jako podmiot nadrzędny lub podrzędny (w pracy GPIF II podlega urządzeniu głównemu, czyli FPGA).

Posiadane funkcje:

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

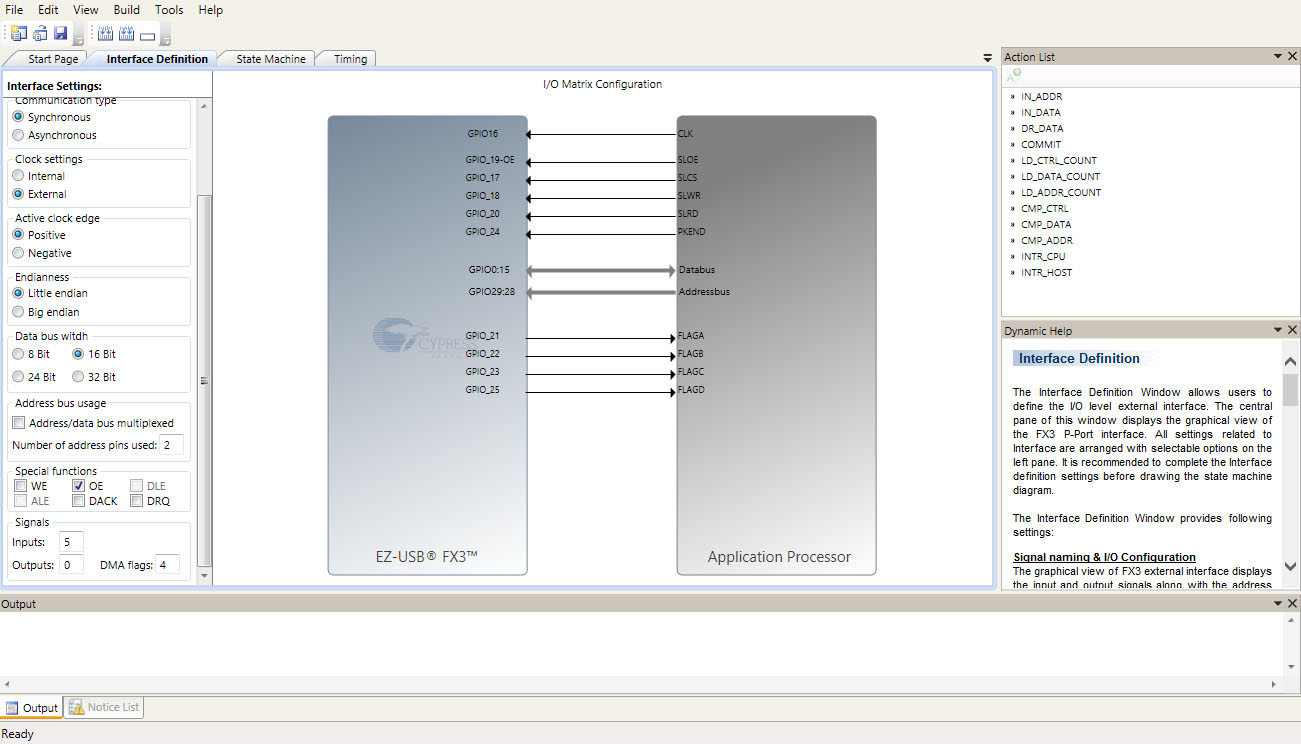
Zachowanie maszyny stanów jest zdefiniowane w deskryptorze, składa się on głównie z zestawu programowalnych rejestrów konfiguracyjnych. W kontrolerze EZ-USB FX3 została dla niego przydzielona pamięć 8 kB.

## 3.4. Narzędzie GPIF II Designer

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

### 3.4.1. Utworzenie i edycja sygnałów interfejsu

Rozpoczęto od importu projektu *sync\_slave\_fifo\_2bit* przygotowanego przez firmę Cypress. Jednak w tym przypadku nie ma możliwości zmiany najważniejszych parametrów interfejsu, więc należy zapisać projekt jako edytowalny (*File - Save Project As Editable*). Po otworzeniu właściwego projektu ukazują się nam pełne możliwości edycyjne programu GPIF II Designer.



Rys. . Okno programu GPIF II Designer

W lewej części okna wybierając odpowiednie opcje można dostosować projekt do własnych potrzeb (Rys. 8). Przeprowadzone zmiany:

* typ interfejsu: podrzędny (ang. slave)
* komunikacja: synchroniczna
* zegar: 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 należało zmienić domyślne nazwy na docelowe (np. SLCS, FLAGA) dla zwiększenia czytelności projektu. Każdy sygnał może mieć osobną konfigurację oraz przypisany inny pin na procesorze FX3. Przyjęto poniższe ustawienia (Tab. 3):

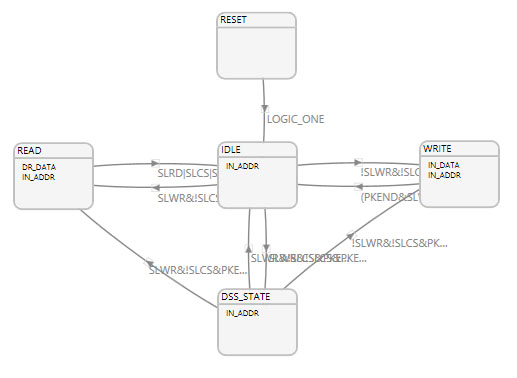
Tab. . Konfiguracja sygnałów interfejsu

|  |  |  |
| --- | --- | --- |
| **Nazwa sygnału** | **Logiczne "1"** | **Pin FX3** |
| CLK | Stan niski | 16 |
| SLOE | 19 |
| SLCS | 17 |
| SLWR | 18 |
| SLRD | 20 |
| PKTEND | 24 |
| Szyna danych | Nie dotyczy | 0:15 |
| Szyna adresowa | Nie dotyczy | 29:28 |

### 

### 3.4.2. Edycja maszyny stanów

Standardowo maszyna stanów obejmowała 7 stanów, zmniejszono tę liczbę do 5 stanów (Rys. 9).



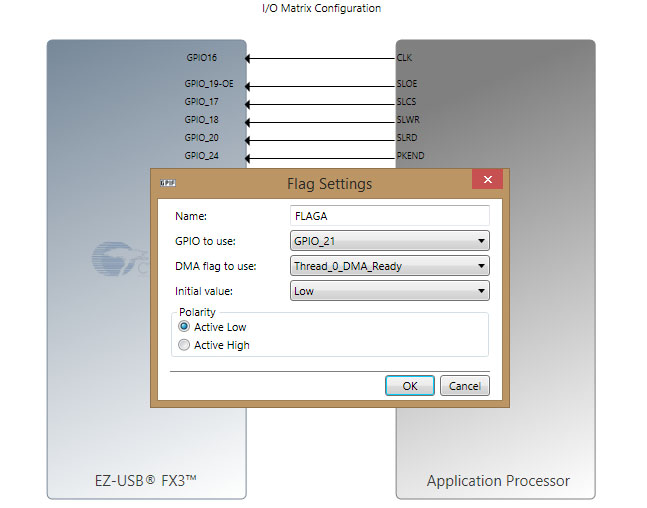
Rys. . Maszyna stanów programowalnej maszyny stanów GPIF II

Można ją utworzyć od podstaw, jednakże oparto się na projekcie, który zaimplementowała firma Cypress.

### 3.4.3. Edycja flag pełnych oraz częściowych

W tym miejscu została przedstawiona jedynie praktyczna strona realizacji flag pełnych i częściowych. Niezbędna teoria jest zawarta w następnym dziale poświęconym urządzeniu FX3.

Wykorzystano 4 gniazda, zatem były potrzebne 4 flagi. Dwie z nich (A i C) są flagami pełnymi, a B i D flagami częściowymi.



Rys. . Przyporządkowanie flagi do wątku

Na nazwie flagi należy wybrać kursorem ustawienia sygnału (*DMA Flag Settings*) oraz przyporządkować jej właściwy wątek (Rys. 10). Tab. 4 przedstawia konfigurację wszystkich czterech flag:

Tab. . Konfiguracja flag w GPIF II Designerze

|  |  |  |  |
| --- | --- | --- | --- |
| **Pin 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 wszelkich modyfikacji zapisano projekt oraz dokonano jego skompilowania. To działanie miało na celu wygenerowanie w folderze projektu pliku *cyfxgpif2config.h* zawierającego konfigurację programowalnej maszyny stanów.

# 4. Cypress EZ-USB FX3

Urządzenie EZ-USB FX3 (w tej pracy często skraca się nazwę do FX3) firmy Cypress jest kontrolerem USB, który umożliwia dodanie funkcjonalności magistrali USB 3.0 do dowolnego procesora lub układu (Rys. 11). Procesor FX3 został oparty na architekturze ARM9. Oferuje on łączność z zewnętrznymi urządzeniami poprzez magistrale zarówno szeregowe jak i równoległe.

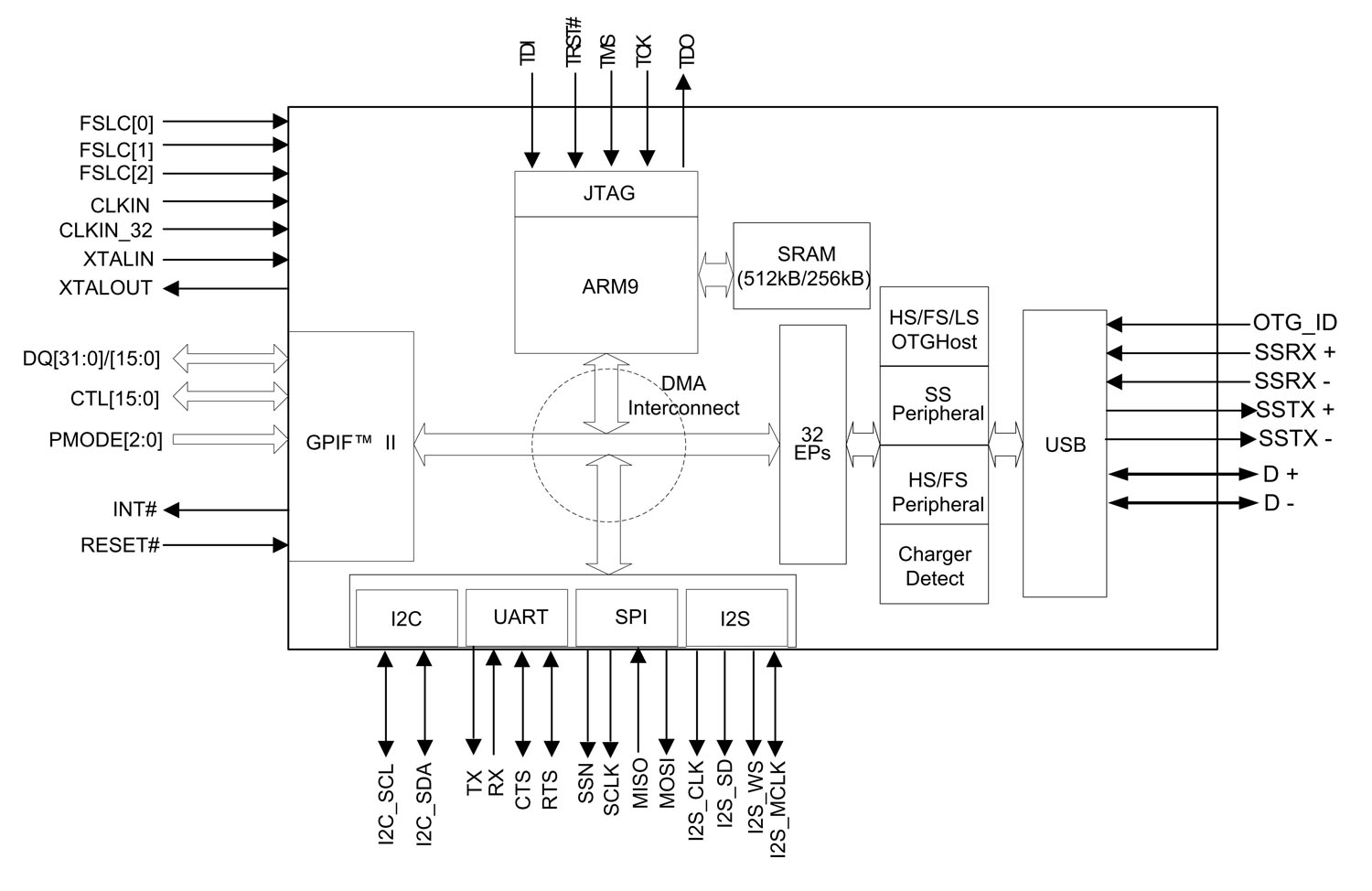


Rys. . Zestaw Cypress EZ-USB FX3 (CYUSB3KIT-001)

: Główną funkcją FX3 jest przesył danych o wysokiej przepustowości pomiędzy hostem USB (komputerem), a urządzeniem zewnętrznym, np. FPGA, kamerą czy skanerem. Moc procesora na rdzeniu ARM9 pozwala również na manipulację danymi bez większych obciążeń. Jednakże w tej pracy nie korzysta się z tych możliwości, gdyż celem nadrzędnym jest dokładna transmisja danych bez jakichkolwiek ingerencji w pakiety.

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 Cypressa). Dzięki niej możliwe jest podłączenie do FX3 układów programowalnych bez względu na producenta (Xilinx, Altera).

Schemat budowy FX3 (Rys. 12) zaczerpnięty z dokumentacji producenta



Rys. . Schemat blokowy FX3

Opis opcje 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 endpointów
* GPIF II, szerzej omówione w rozdziale 3.3
* 32-bitowy procesor ARM926EJ-S oparty na rdzeniu ARM9, 512 kB pamięci SRAM
* interfejs JTAG - 5-pinowy 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 bitów/s do 4608 Kb/s
* I2C - FX3 działa jako urządzenie nadrzedne (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 - wspierana jest magistrala operująca na maksymalnej częstotliwości 33 MHz
* zasilanie - każdy blok peryferyjny może być zasilany z osobnego źródła o różnym napięciu
* zegar - 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

Rozdział ma za zadanie przybliżyć teoretyczną stronę realizacji synchronicznej kolejki FIFO. Do tego niezbędne jest zaznajomienie się z poniższymi terminami.



Rys. . Schemat działania gniazd oraz wątków. W jednym momencie z FPGA może współpracować tylko jeden wątek, reszta jest nieczynna

### 4.1.1. Gniazda

Gniazdo jest punktem połączeniowym pomiędzy blokiem peryferyjnym oraz pamięcią RAM FX3 (Rys. 13). Każdy blok (np. USB, GPIF, UART czy SPI) posiada określoną liczbę gniazd z nim związanych. Liczba gniazd odpowiada liczbie niezależnych danych przechodzących przez gniazdo (np. jedno gniazdo odpowiadające transferowi danych z USB do GPIF, a drugie z GPIF do USB).

### 4.1.2. Deskryptory DMA

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

### 4.1.3. Bufor DMA

Bufor DMA jest sekcją w pamięci RAM, służy do pośredniego zapisu i przetrzymywania danych przesyłanych dzięki urządzeniu FX3. Bufory są zaalokowane, podobnie jak deskryptory, w pamięci RAM przez oprogramowanie FX3. Ich adresy są przechowywane w deskryptorach DMA.

### 4.1.4. Wątki GPIF II

Pojedynczy wątek GPIF II to dedykowana ścieżka danych w bloku maszyny stanów, który łączy zewnętrzne peryferia z GPIF-em poprzez gniazda (Rys. 13). 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 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 jest nazywane producentem. Z kolei gniazdo, które z niego odczytuje jest 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 koreluje z nazwami gniazd, np. gniazdo 0 - wątek 0, gniazdo 4 - wątek 4.

Linia adresowa (A1:A0) interfejsu synchronicznej kolejki FIFO wskazuje na wątek, który ma być aktualnie używany. DMA jest tak przygotowane przez firmę Cypress, że zmiana wątku automatycznie pociąga za sobą zmianę gniazda, zatem jeśli zmieniono adres z 0 na 3 to mapowane jest gniazdo 3 do wątku o numerze 3.

Gniazda, do których chce się uzyskiwać dostęp najpierw muszą być określone w konfiguracji kanału DMA, o czym traktuje kolejny podrozdział.

### 4.1.5. Konfiguracja kanału DMA

Oprogramowanie służy do skonfigurowania kanału DMA przypisując do niego gniazda producenckie (źródła danych) oraz konsumenckie (pobierające dane).

**Przykład**: jeśli dane są przesyłane z bloku GPIF do USB to wtedy blok GPIF jest producentem, a USB konsumentem.

Zatem jeśli dane są przesyłane 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 wskazywała zapełnienie bufora, gdy 1024 bajtów danych zostanie zapisanych do pierwszego bufora. Dalej flaga będzie wskazywała 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 (master, w pracy FPGA) musi ciągle monitorować stan flagi, by wiedzieć kiedy zakończono przełączanie się do pustego buforu.

## 4.2. Konfiguracja flag pełnych oraz częściowych

Flagi mogą być skonfigurowane jako puste, pełne, częściowo puste lub częściowo pełne sygnały. Nie są one sterowane przez programowalną maszynę stanów GPIF II, lecz przez 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 transferuje dane. Zatem flaga wskazuje stan pusty / niepusty jeśli dane są odczytywane z gniazda lub status pełny / niepełny 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 trzeba uwzględnić w oprogramowaniu.

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

W tym trybie flaga została tak skonfigurowana, aby wskazywała 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 zmienia się wątek (linia adresowa zmienia się np. z 0 na 3).

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

Opóźnienia związane z flagami podczas odczytu / zapisu danych są umieszczone w Tab. 5.

### 4.2.2. Flagi wskazujące na aktualny wątek

Flagi skonfigurowane w ten sposób 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. Ta konfiguracja nie angażuje dodatkowych linii w interfejsie synchronicznej kolejki FIFO, wystarczy 1 flaga do poprawnego pokazywania stanów 4 wątków. Wprowadza to jednak opóźnienie w postaci 2 taktów zegara, ponieważ GPIF najpierw musi sprawdzić stan na linii adresowej, a dopiero potem zaktualizować flagę.

### 4.2.3. Flaga częściowa

Jest jeszcze jedna opcja konfiguracji - flaga częściowa. Wskazuje ona status częściowo pustego / pełnego gniazda. Dodatkowo należy obliczyć wartość liczby (ang. watermark) niezbędnej do skonfigurowania gniazda używając jednej z funkcji API Cypressa.

Poniższa Tab. 5 przedstawia opóźnienia, które są generowane 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

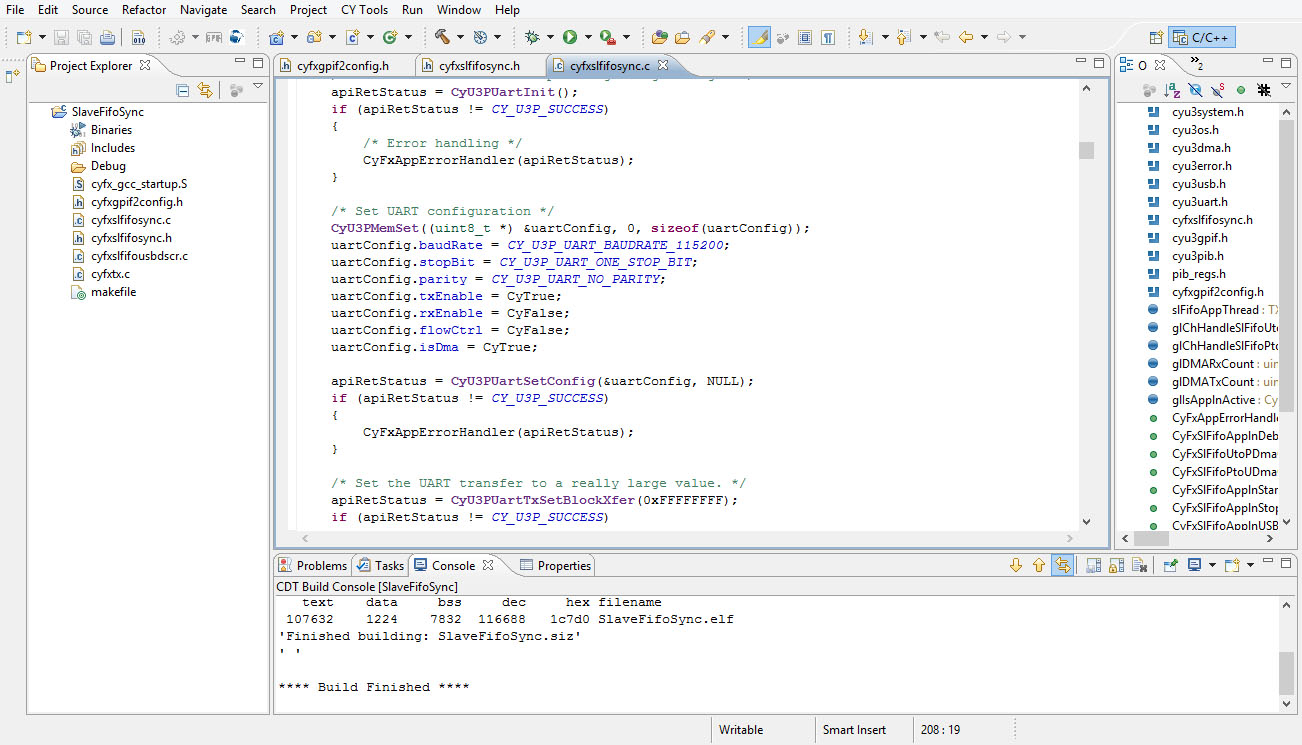
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Typ flagi** | **Opcja w programie GPIF II Designer** | **Opóźnienie związane z adresowaniem** | **Opóźnienie flag na koniec transferu** | | **Wykorzystanie API** |
| **Zapis danych** | **Odczyt danych** |
| Pełna flaga dedykowana do wątku | 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 program przykładowy, który posłużył jako baza do zrealizowania interfejsu synchronicznej kolejki FIFO.

### 4.3.1. Zintegrowane środowisko programistyczne - 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 (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

Wygląd środowiska nie odbiega od innych IDE (Rys. 14). Po lewej stronie znajduje się drzewo plików projektu, w środku karty z otwartymi plikami źródłowymi, po prawej zmienne i funkcje, a 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 ciągłe wysyłanie lub odbieranie danych, a drugi obsługujący pętlę. Jest to spowodowane tym, że oba projekty posiadają różną konfigurację oraz różne stałe związane z transferem danych.

### 4.3.2. Ciągłe wysyłanie lub odbieranie danych

Pierwszym krokiem było dołączenie pliku cyfxgpif2config.h do projektu, który został 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. 6), które służą do konfiguracji ustawień magistrali USB podczas wykorzystania funkcji *CyU3PDmaChannelCreate, CyU3PThreadCreate, CyU3PDmaChannelSetXfer* oraz struktury *CyU3PDmaChannelConfig\_t*. Oto one:

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 | jw. |
| 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 | jw. |
| CY\_FX\_SLFIFO\_THREAD\_STACK | 0x04 | Wielkość stosu wątku |
| CY\_FX\_SLFIFO\_THREAD\_PRIORITY | 8 | Priorytet wątku |
| CY\_FX\_EP\_PRODUCER | 0x01 | Numer endpointu |
| CY\_FX\_EP\_CONSUMER | 0x81 | jw. |
| 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 przedstawiano modyfikacje przeprowadzone w głównym kodzie źródłowym (plik *cyfxslfifosync.c*):

* dołączenie pliku *cyfxgpif2config.h* dyrektywą preprocesora include
* zadeklarowanie zmiennej *burst\_length*, która odpowiada za liczbę pakietów podczas jednego transferu oraz przypisanie jej odpowiednich wartości dla różnych prędkości USB. Następnie zmienna wykorzystywana w przypisaniu wartości jednemu z elementów struktury *CyU3PEpConfig\_t*, która jest drugim argumentem funkcji *CyU3PSetEpConfig* do konfiguracji endpointu
* utworzenie automatycznego lub manualnego kanału DMA, wykorzystanie stałych z pliku nagłówkowego oraz funkcji *CyU3PDmaChannelCreate*
* utworzenie funkcji GPIFErrorCallback służącej do debugowania programu, komunikaty są wysyłane do komputera przy użyciu interfejsu UART
* konfiguracja programowalnej maszyny stanów przy użyciu funkcji *CyU3PGpifLoad* oraz danych wygenerowanych przez narzędzie GPIF II Designer
* wybranie, konfiguracja oraz aktywacja gniazda GPIF, użycie funkcji *CyU3PGpifSocketConfigure* jedynie jeśli wcześniej skonfigurowano flagi częściowe
* uruchomienie programowalnej maszyny stanów GPIF umożliwia funkcja *CyU3PGpifSMStart*, jej argumenty to stan początkowy maszyny stanów
* rejestracja powiadomień podczas przerwań za pomocą *CyU3PPibRegisterCallback*
* ustawiono pętlę PLL, dzięki której FX3 będzie taktowany zegarem o częstotliwości 400 MHz, odpowiada za to linia *clockConfig.setSysClk400 = CyTrue*
* zarządzanie pamięcią podręczną odbywa się przez podanie odpowiednich argumentów do funkcji *CyU3PDeviceCacheControl* (pamięć podręczna jest czyszczona przez użytkownika)

### 4.3.3. Pętla

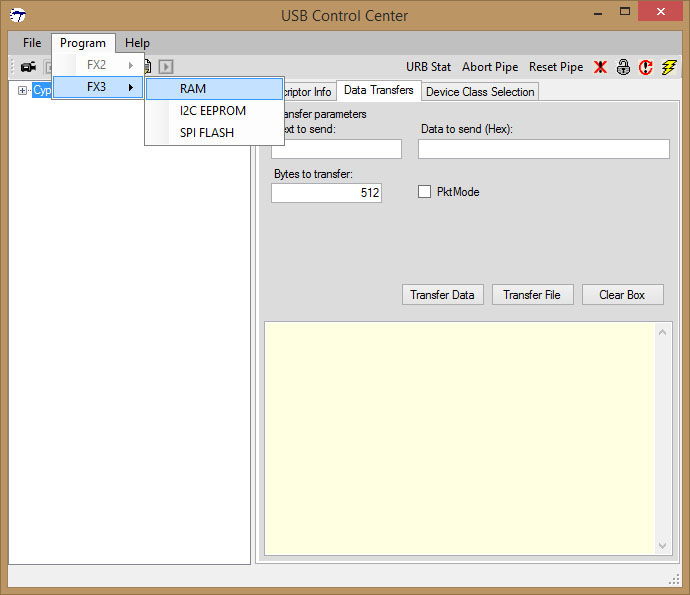
Projekt realizujący pętlę nie odbiega zasadniczo od poprzedniego programu, również za bazę posłużyły pliki przygotowane przez firmę Cypress. Większość kodu jest identyczna, zmiany zostały przeprowadzone w następujących miejscach:

* zmiana wartości stałej *DMA\_BUF\_SIZE* z 16 na 1
* zmiana 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
* przypisanie wartości "1" zmiennej globalnej *glIsApplnActive* odpowiadającej za wykonywanie pętli (w przypadku ciągłego odbierania / wysyłania miała ona wartość "0")

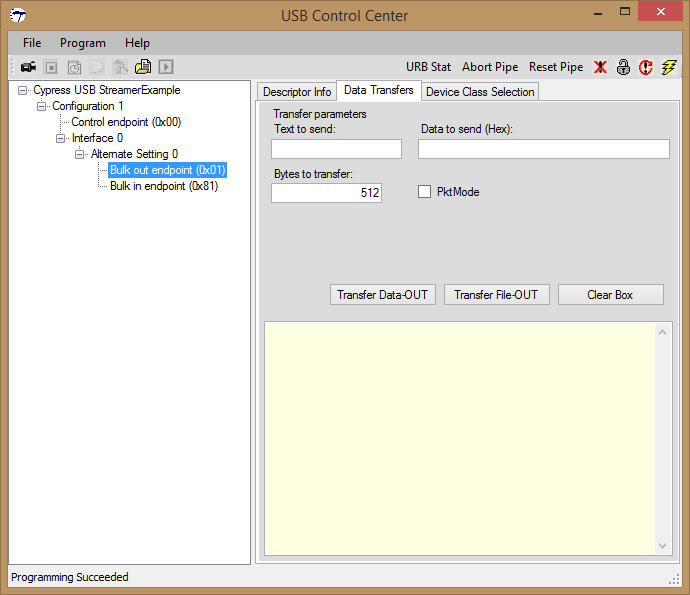
## 4.4. USB Control Center - programowanie układu

Po kompilacji układu otrzymano plik o rozszerzeniu .img, który należy załadować do pamięci RAM FX3 w celu zaprogramowania układu.

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. 15). Należało wskazać plik .img na dysku komputera, po czym nastąpiło wgranie programu do pamięci procesora. Zmieniła się nazwa urządzenia na Cypress USB Streamer, zatem można rozwinąć drzewo konfiguracji oraz uzyskać dostęp do wysyłania lub odbierania danych w konkretnych endpointach (Rys. 16).



Rys. . Wybór pliku .img do zaprogramowania układu



Rys. . Po zaprogramowaniu, uzyskany dostęp do endpointów

# 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 jeśli chodzi o zaawansowanie układu. Poniżej przedstawiono najważniejsze cechy układu oraz płytki ewaluacyjnej:

* standard logiczny: 3,3 V
* moduł Cyfrowego Zarządzania Zegarem - 4 bloki
* maksymalna ilość wejść/wyjść - 232
* możliwość konfiguracji po załączeniu zasilania za pomocą Xilinx 4 Mbit Flash PROM
* zewnętrzny zegar 50 MHz
* złącze Hirose FX2 do szybkiej transmisji danych
* cztery przełączniki wykorzystywane do zmiany trybu transmisji
* wyświetlacz LCD ze sterownikiem HD44780 działający 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 na stronie producenta. Jest to najnowsza wersja środowiska programistycznego udostępniona 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 pinów wejściowych/wyjściowych) czy Xilinx Impact (programowanie układu docelowego lub układu wykorzystującego plik PROM do wstępnej konfiguracji FPGA).

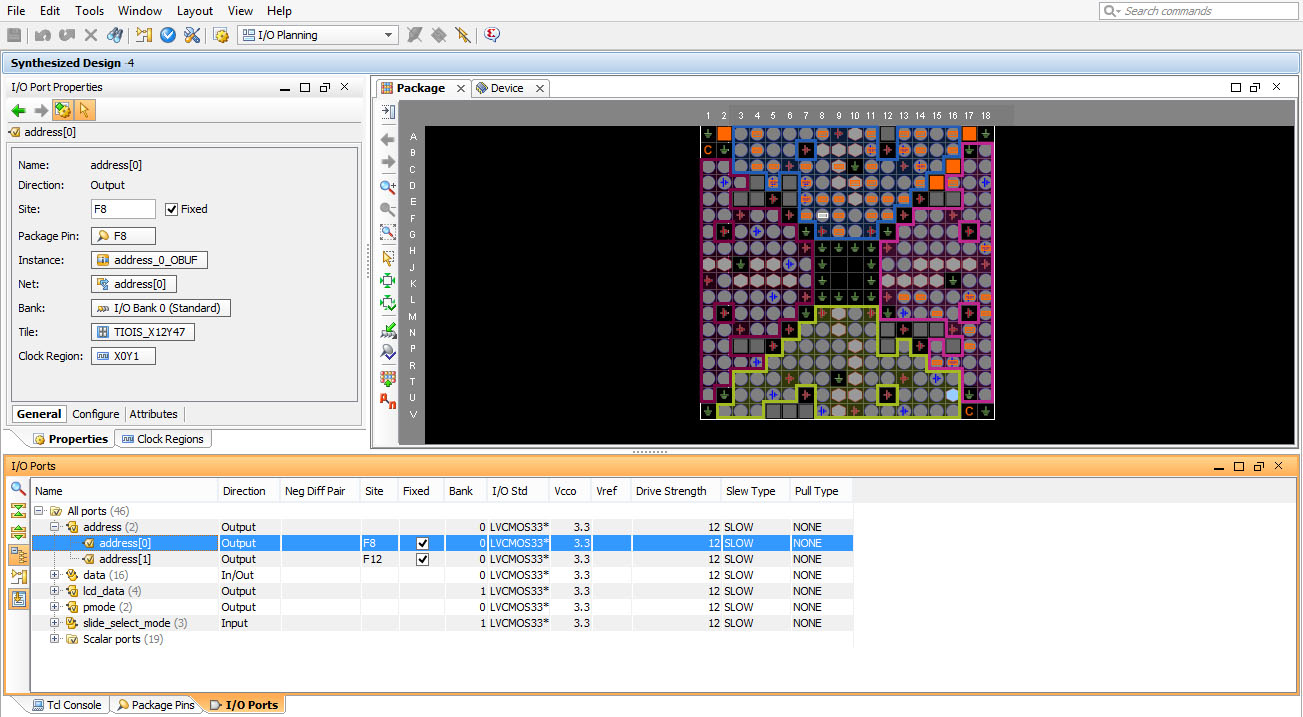
## 5.2. Utworzenie projektu, przypisanie pinów

Nowy projekt utworzono 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 pinów do zadeklarowanych portów układu. Posłużono się modułem Xilinx PlanAhead (Rys. 17).



Rys. 17. Xilinx FlanAhead - widok programu

W części *I/O Ports* są dostępne wszystkie wejścia i wyjścia zdefiniowane w części deklaracyjnej projektu (ang. entity). Wybierając konkretny sygnał można w zakładce *I/O Port Properties* dokonano przypisania sygnału. Po zaznaczeniu wszystkich sygnałów (*All ports*) i kliknięciu prawym przyciskiem myszy wybrano opcję *Configure I/O Ports*. Zmieniono domyślny standard z LVCMOS18 na LVCMOS33 (do portów doprowadzone jest znamionowe napięcie 3,3 V), moc sygnału z 4 na 12. Jedynie porty odpowiadające sygnałom dostępnym na złączu Hirose FX2 (wykorzystywane do komunikacji z kontrolerem USB) posiadają szybką prędkość (ang. fast), reszta portów działa w standardowym wolnym trybie (ang. slow). Po zapisaniu ustawień moduł automatycznie wygeneruje plik .ucf, który jest niezbędny do syntezy projektu. Poniżej zamieszczono Tab. 7 z przypisaniami każdego użytego pinu.

Tab. . Przypisania pinów

|  |  |  |
| --- | --- | --- |
| **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 | jw. |
| slide\_select\_mode[2] | H18 | jw. |
| led\_buffer\_empty\_show | F9 | Status pamięci FIFO |
| lcd\_e | M18 | Sygnał Enable LCD |
| lcd\_rs | L18 | Sygnał Register Select LCD |
| lcd\_rw | L17 | Sygnal Read/Write LCD |
| lcd\_srataflash\_disable | D16 | Wyłączenie modułu FLASH LCD |
| lcd\_data[0] | R15 | 4-bitowa linia danych |
| lcd\_data[1] | R16 | jw. |
| lcd\_data[2] | P17 | jw. |
| lcd\_data[3] | M15 | jw. |
| clock100\_out | D10 | Sygnał zegarowy do FX3 |
| flaga | A13 | Flaga inf. o stanie buforu FX3 |
| flagb | C5 | jw. |
| flagc | E7 | jw. |
| flagd | F7 | jw. |
| 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 | jw. |
| data[0] | A14 | Linia danych |
| data[1] | C14 | jw. |
| data[2] | D14 | jw. |
| data[3] | B14 | jw. |
| data[4] | E11 | jw. |
| data[5] | A16 | jw. |
| data[6] | E9 | jw. |
| data[7] | C11 | jw. |
| data[8] | B13 | jw. |
| data[9] | G9 | jw. |
| data[10] | B16 | jw. |
| data[11] | B11 | jw. |
| data[12] | C4 | jw. |
| data[13] | A11 | jw. |
| data[14] | E13 | jw. |
| data[15] | A8 | jw. |
| pmode[0] | D11 | Sygnały do debugowania |
| pmode[1] | F11 | jw. |
| reset\_to\_fx3 | E8 | Sygnał resetujący FX3z FPGA |
| reset\_from\_fx3 | E12 | Sygnał resetujący FPGA z FX3 |

## 5.3. Cyfrowe Zarządzanie Zegarem (DCM - Digital Clock Manager)

FPGA jest głównym układem (ang. master) w interfejsie synchronicznej kolejki FIFO, więc kontroler EZ-USB FX3 jest układem podrzędnym (ang. slave). Aby przesyłanie danych przebiegało bezproblemowo firma Cypress zaleca następującą konfigurację:

* EZ-USB FX3 powinien być taktowany zegarem o częstotliwości 400 MHz, pociąga to za sobą skonfigurowanie pętli PLL, aby przekształciła doprowadzony sygnał zegarowy 19.2 MHz do docelowej wartości
* doprowadzenie do programowalnej maszyny stanów GPIF II sygnału zegarowego z układu głównego (FPGA) o częstotliwości 100 MHz

Pierwszy podpunkt wykonuje się za pomocą kilku instrukcji wykorzystując API Cypressa (szerzej jest to omówione w poprzednim rozdziale). Natomiast uzyskanie 100 MHz z 50 MHz (tyle wynosi częstotliwość zegara na płytce uruchomieniowej FPGA) wiąże się z wykorzystaniem gotowego modułu przygotowanego przez firmę Xilinx, a mianowicie DCM (Cyfrowe Zarządzanie Zegarem, and. Digital Clock Manager).

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

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

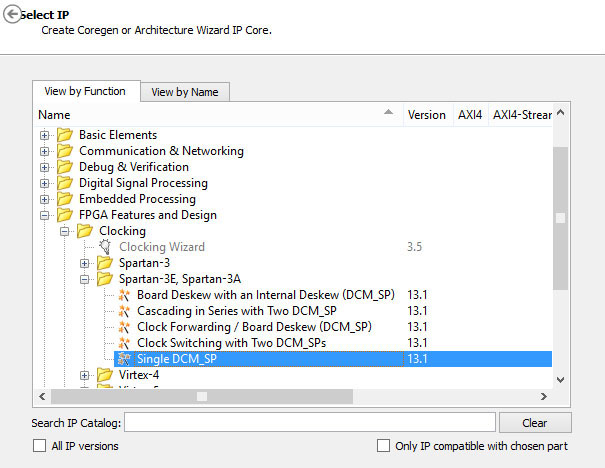
Celem wykorzystania modułu DCM było uzyskanie 100 MHz, a zatem częstotliwości dwa razy większej od oryginalnej. Nie skorzystano zatem z możliwości wyboru mnożników i dzielników, a zaznaczono opcję CLK2FX - znacznie prostsze rozwiązanie. Proces konfiguracji DCM (Rys. 18 oraz Rys. 19):

* dodanie nowego źródła w drzewie projektu
* wybór opcji IP (CORE Generator & Architecture Wizard)
* drzewo opcji: FPGA Features and Design - Clocking - Spartan 3E - Single DCM\_SP
* podanie wartości "50 MHz" w polu Input Clock Frequency, zaznaczenie opcji "zewnętrzny zegar" (CLKIN Source - External)
* zaznaczenie opcji CLK2X oraz RST

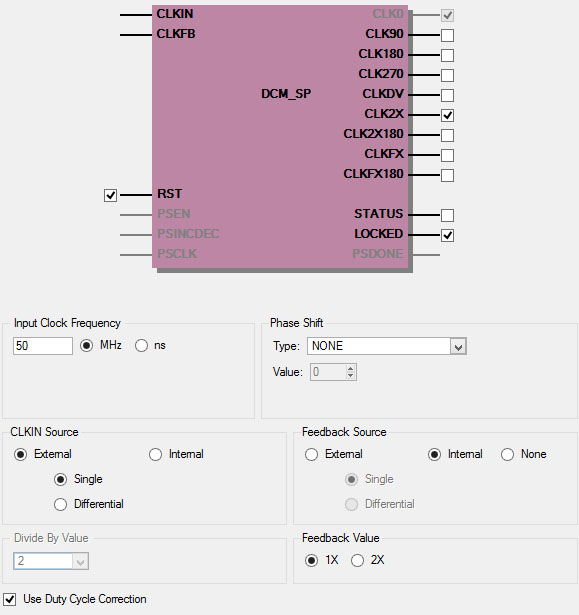
Po wygenerowaniu modułu należy zaznaczyć go i wybrać z listy *View HDL Instantiation Template* w celu uzyskania kodu, który można dołączyć do głównego pliku .vhd. Są to sekcje *component* (deklaracja jednostki projektowej) oraz *port map* (mapowanie sygnałów). W sekcji mapowania dokonujemy przypisania wchodzących oraz wychodzących sygnałów do konkretnego modułu (Tab. 8).

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. 18. Wybór DCM z listy IP Cores



Rys. 19. Konfiguracja bloku DCM

## 5.4. Pamięć FIFO

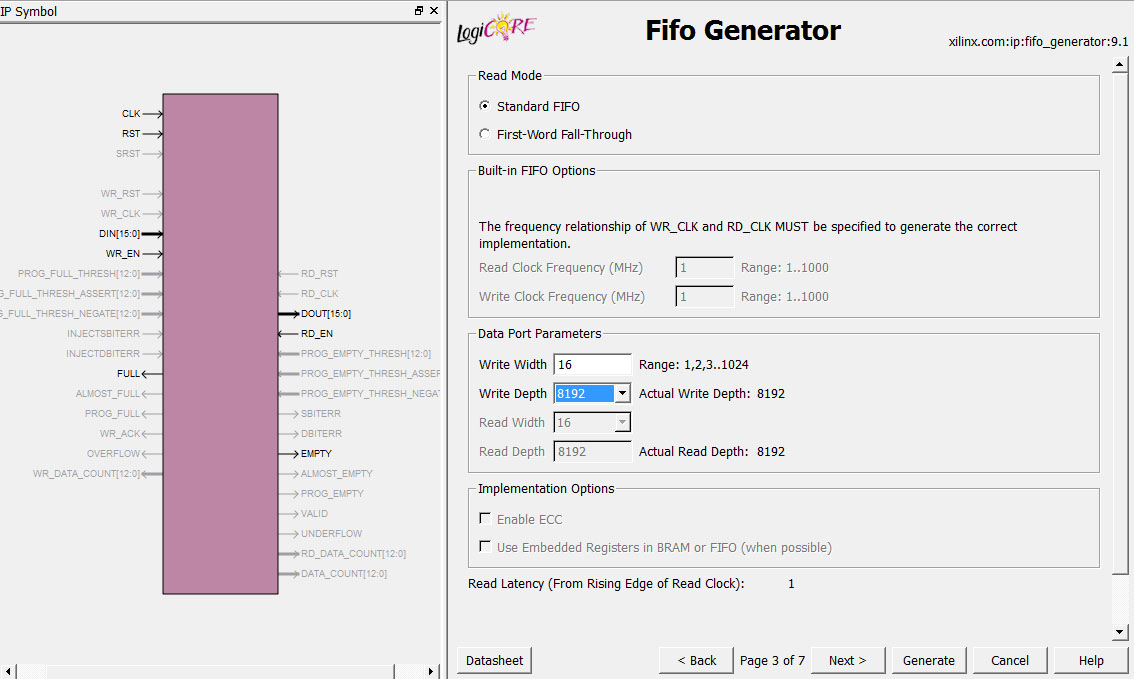
W celu implementacji pętli (ang. Loopback Transfer, czyli wysyłanie i odbiór identycznych pakietów danych w komputerze) niezbędna jest pamięć FIFO (ang. First In, First Out). Posłużono się modułem *Fifo Generator* zrealizowanym przez firmę Xilinx dostępnym (podobnie jak DCM) w katalogu IP Core. Konfiguracja (Rys. 20) przebiegła następująco:

* dodanie nowego źródła w drzewie projektu, wybranie *IP Cores* oraz podanie nazwy modułu
* wybranie *Memories & Storage Elements - Fifo Generator* w wersji 9.1
* port wejściowy oraz wyjściowy mają identyczną pojemność: szerokość 16 bitów, głębokość 8192 bitów

Po zatwierdzeniu ustawień postąpiono analogicznie jak w przypadku DCM-a, czyli skopiowano do pliku zawierającego transfer pętli wygenerowany w IDE kod. Sygnały modułu (Tab. 9):

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 na dane |
| 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. 20. Konfiguracja pamięci FIFO

## 5.5. Ogólna struktura programu

Program napisano w języku opisu sprzętu VHDL. Został on podzielony na indywidualne moduły, każdy z nich jest maszyną o skończonej maszynie stanów:

* slave\_fifo\_main - moduł główny, zarządzanie sygnałami
* lcd\_controller - obsługa wyświetlacza LCD
* slave\_fifo\_stream\_write\_to\_fx3 - ciągłe wysyłanie danych do PC Z FPGA
* slave\_fifo\_stream\_read\_from\_fx3 - ciągłe wysyłanie danych do FPGA z PC
* slave\_fifo\_loopback - pętla

Jak zostało wcześniej nadmienione, doprowadzamy sygnał zegarowy o częstotliwości 50 MHz. Wybór transmisji wybieramy za pomocą trzech przełączników umiejscowionych po prawej stronie wyświetlacza LCD. Czwarty przełącznik służy do resetowania układu. Poniżej przedstawiono tabelę z możliwymi ustawieniami przełączników (Tab. 10).

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 |

Wyświetlacz LCD informuje użytkownika, w którym module obecnie się znajduje. Ponadto dla trzech trybów również wyświetla odbierane/wysyłane dane (16 bitów).

Kolejne podrozdziały mają za zadanie zaprezentować sposób działania każdego modułu.

### 5.5.1. Główny program - Slave Fifo Main

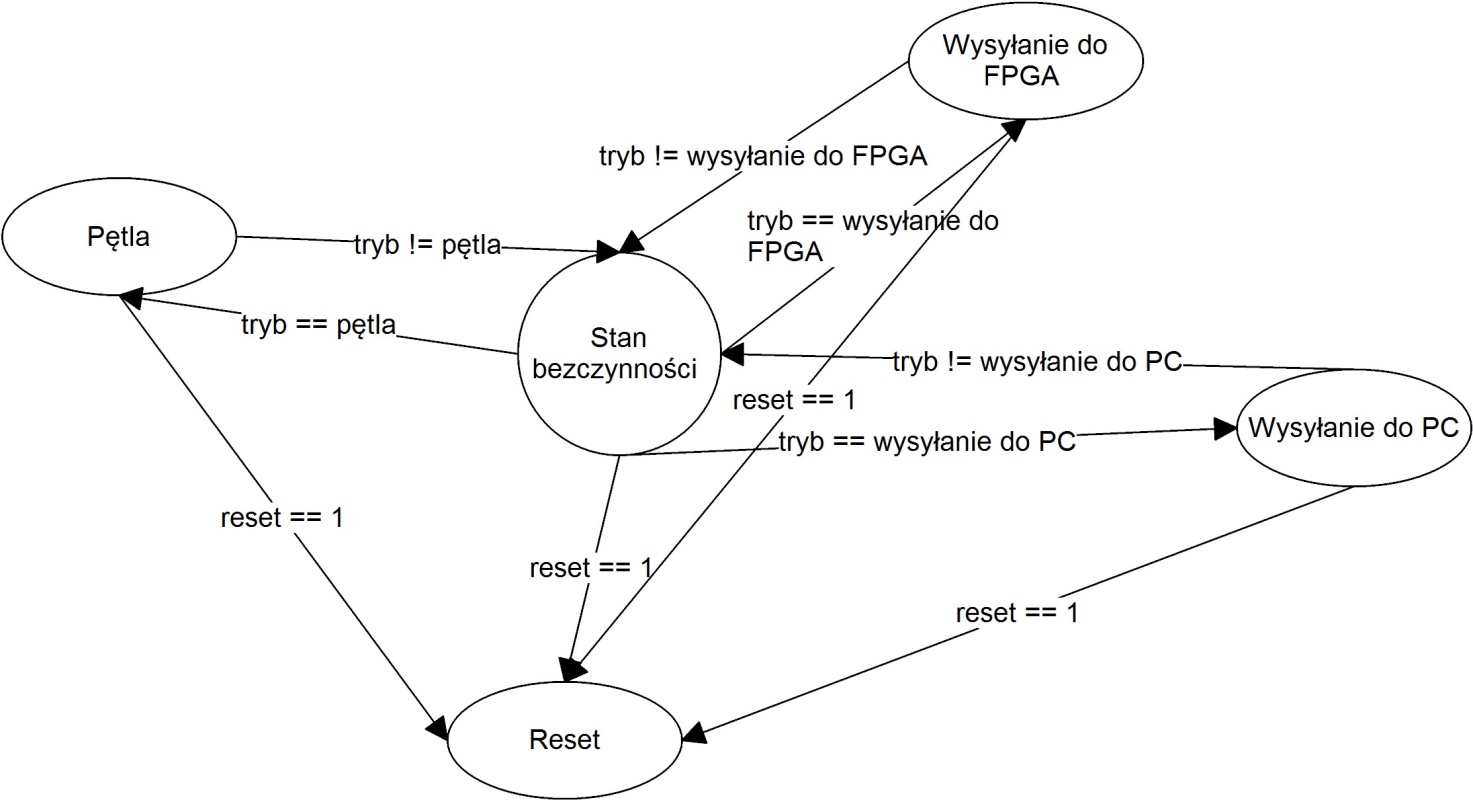
Główny program zaczyna się od deklaracji jednostki (entity). Zdefiniowane są wszystkie sygnały wyjściowe i wejściowe (opisane wcześniej w Tab. 7). W architekturze zdefiniowane są stałe, pomocnicze sygnały oraz zmienne maszyny stanów. Następnie zostały zmapowane wszystkie komponenty. Napisano funkcję *vector\_to\_string*, która ma za zadanie konwersję zmiennej typu *std\_logic\_vector* na *string*. Jest ona wykorzystywana do konwersji wchodzących lub wychodzących danych na format umożliwiający wyświetlanie dwóch bajtów w formie pojedynczych bitów 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 odpowiadają odbieraniu lub wysyłaniu danych i monitorowaniu flag maszyny stanów GPIF II. Na końcu programu umieszczono główną maszynę stanów odpowiadającą za przechodzenie do wybranego trybu transmisji.

Stany (zmienne) głównego programu:

* idle\_state (stan bezczynności)
* loopback\_state (pętla)
* stream\_read\_from\_fx3\_state (ciągłe wysyłanie danych do FPGA)
* stream\_write\_to\_fx3\_state (ciągłe wysyłanie danych do PC)

Schemat graficzny przedstawia zasadę działania (Rys. 21).



Rys. 21. Maszyna stanów głównego programu

Analizując powyższy diagram można dojść do wniosku, że nie można przejść z trybu np. ciągłego wysyłania danych z PC do FPGA do pętli pomijając stan bezczynności (idle). Adres aktualnego wątku jest zależny 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 znajduje się jeszcze obsługa ekranu LCD, która została również podzielona na poszczególne etapy, a co za tym idzie - użyto maszyny stanów. Oto one:

* 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)

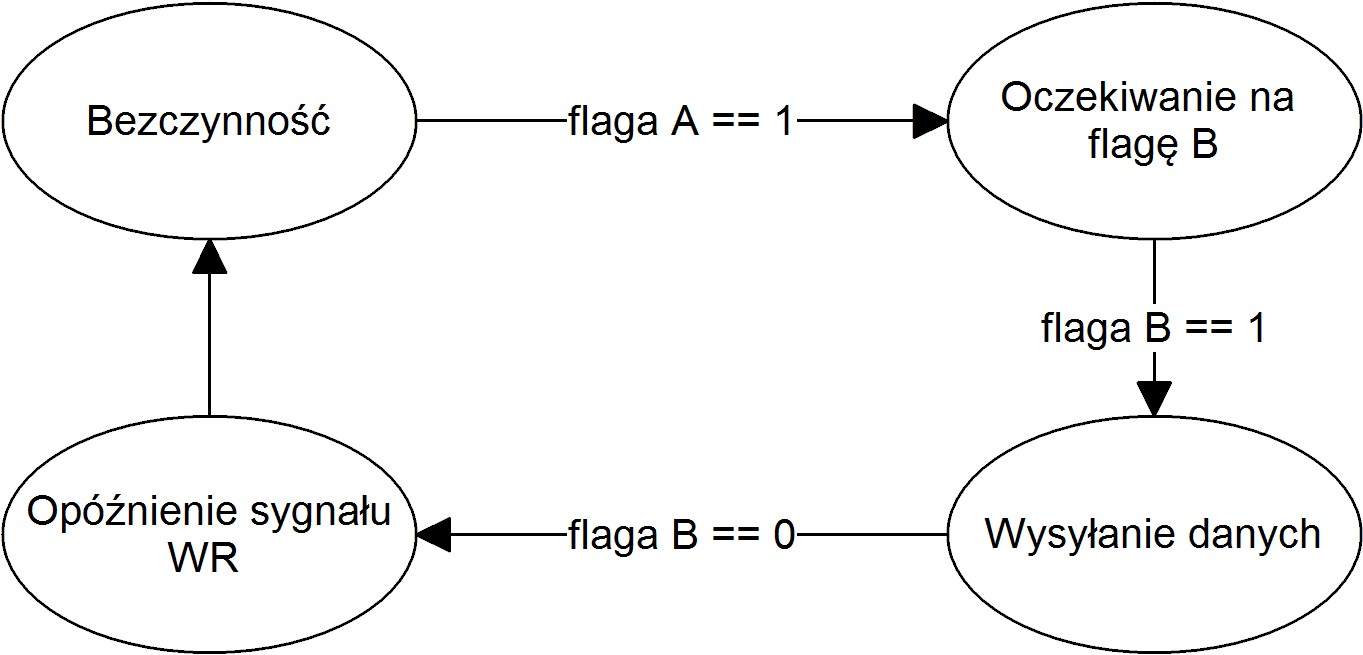
Za dokładną obsługę 4-bitowej linii danych oraz pozostałych sygnałów ekranu LCD odpowiada moduł *lcd\_controller*.

### 5.5.2. Ciągłe wysyłanie danych do komputera - Stream Write to FX3

Po deklaracji jednostki projektowej, stałych i sygnałach pomocniczych znajduje się deklaracja maszyny stanów odpowiadająca za poprawne działanie trybu ciągłej wysyłki danych do komputera.

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

Działanie prezentuje poniższy diagram (Rys. 22):



Rys. 22. Schemat działania ciągłego zapisu do PC

Maszyna stanów odpowiada za monitorowanie sygnałów flag pochodzących z kontrolera USB (dostępność buforów DMA) oraz zmienia stany sygnałów związanych z transmisją danych.

W tym trybie manipuluje się jedynie stanem slwr, czyli sygnałem odpowiedzialnym za zapis danych.

* slcs = 0 (wartość stała)
* sloe = 1 (wartość stała)
* slrd = 1 (wartość stała)
* slwr = 0 gdy znajdujemy się w stanie stream\_in\_write oraz flaga B jest równa 1, w innych przypadkach slwr = 1

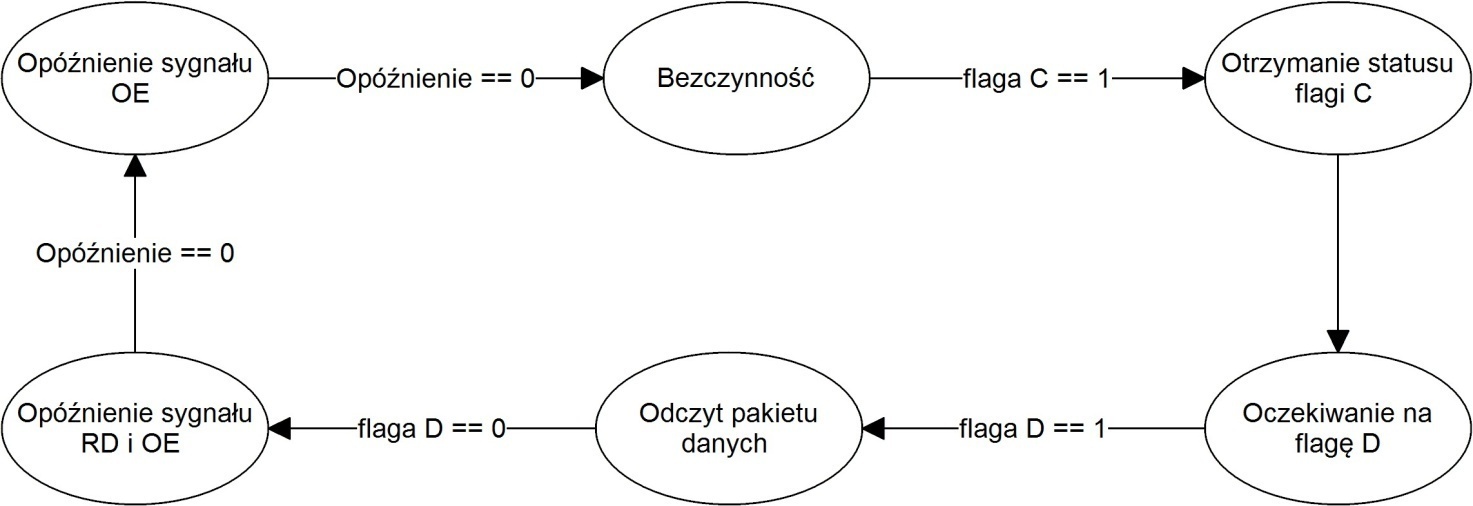
Postanowiono, że program będzie wysyłał do komputera identyczny pakiet danych zapisany w formacie bitowym: 0101011101001101. Gdy przekształci się tę liczbę na format szesnastkowy otrzyma się 0x574D. Liczba 0x57 odpowiada literze "W", a 0x4D literze "M" w kodzie ASCII.

### 5.5.3. Ciągłe odbieranie danych do komputera - Stream Read from FX3

Analogicznie do poprzednich działów ten tryb ma własną maszynę stanów. Oto poszczególne etapy:

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

Zasadę działania jest zamieszczona w poniższym diagramie (Rys. 23):



Rys. . Schemat ciągłego odbierania danych z PC

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

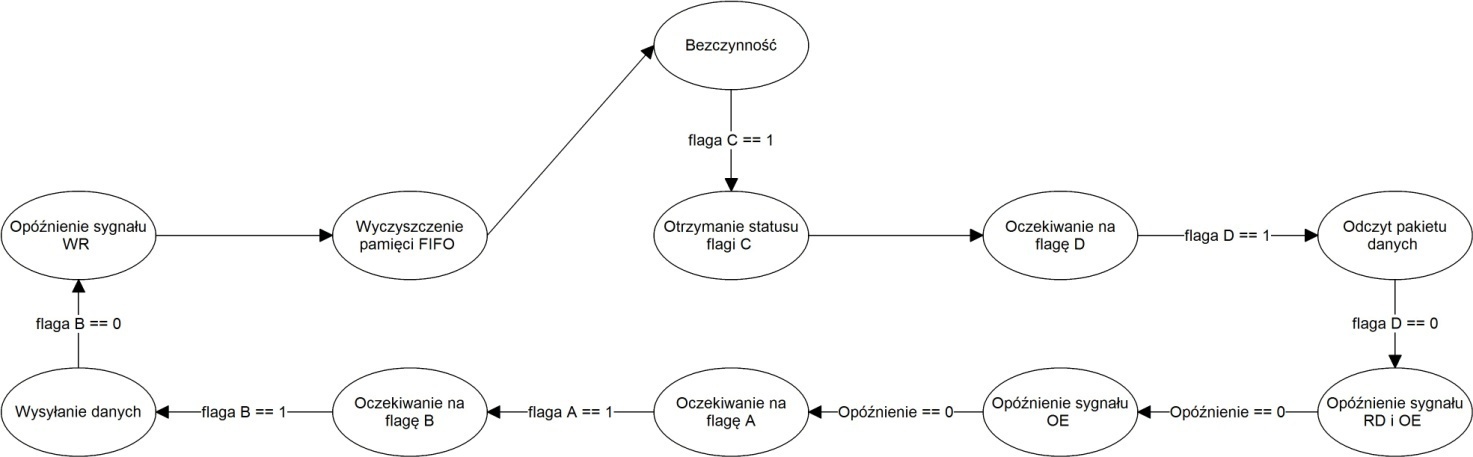
Na osoby akapit zasługuje omówienie opóźnień generowanych po odczycie danych. Zgodnie z zasadami (opisanymi w dokumentacji firmy Cypress odnośnie interfejsu synchronicznej kolejki FIFO) w przypadku użycia częściowych flag jest konieczne, aby FX3 monitorował sygnał Read jeszcze przez kolejne 3 takty zegara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

Przebywając w tym trybie program modyfikuje stany tylko dwóch sygnałów odpowiedzialnych za zapis, są to:

* slcs = 0 (wartość stała)
* slwr = 1 (wartość stała)
* sloe = 0 gdy znajdujemy się w stanie read oraz w obu opóźnieniach, w pozostałych przypadkach sloe = 1
* slrd = 0 gdy znajdujemy się w stanie read oraz w pierwszym opóźnieniu (rd\_oe\_delay), w pozostałych przypadkach slrd = 1

### 5.5.4. Pętla - Loopback Transfer

Ostatni z zaimplementowanych rodzajów przesyłania danych jako jedyny wykorzystuje (omówioną w rozdziale 5.4) pamięć FIFO. Pakiet danych odebrany z magistrali USB jest zapisywany w buforze danych, a następnie przy zwolnieniu się miejsca w kanałach DMA następuje pobranie tego pakietu, wyczyszczenie pamięci oraz odesłanie danych z powrotem do komputera. Pętla jest połączeniem dwóch poprzednich transmisji, zatem diagram jest najbardziej rozbudowany (Rys. 24).



Rys. . Schemat maszyny stanów pętli

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 monitoruje się wszystkie cztery flagi dostępu do buforów DMA oraz manipuluje stanami trzech sygnałów:

* slcs = 0 (wartość stała)
* slwr = 0 gdy znajdujemy się w stanie stream\_in\_write oraz flaga B jest równa 1, w innych przypadkach slwr = 1
* sloe = 0 gdy znajdujemy się w stanie read oraz w obu opóźnieniach, w pozostałych przypadkach sloe = 1
* slrd = 0 gdy znajdujemy się w stanie read oraz w pierwszym opóźnieniu (rd\_oe\_delay), w pozostałych przypadkach slrd = 1

Zasada stosowania opóźnień po odczycie pakietu jest identyczna jak w przypadku transmisji ciągłego odczytu danych.

## 5.6. Programowanie FPGA

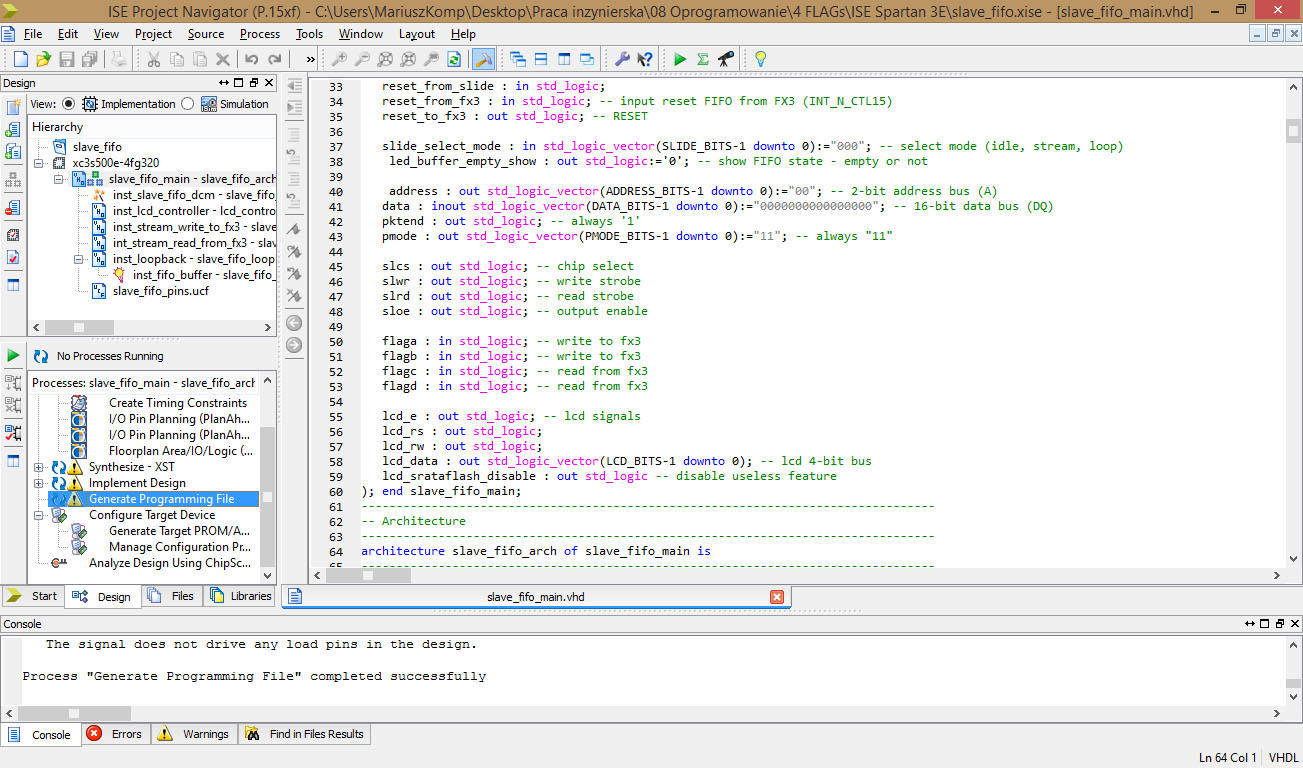
Płytka ewaluacyjna Spartan 3E wspiera cztery możliwości zaprogramowania układu programowalnego:

* bezpośrednia konfiguracja przy wykorzystaniu interfejsu JTAG i wbudowanego na płytce konwertera USB (ISP - 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 pisania kodu oraz jego późniejszego debugowania 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 skończony. Płytka podczas załączenia zasilania pobiera konfigurację z układu XCF04S, nie jest konieczna jakakolwiek ingerencja użytkownika albo potrzeba dalszego programowania.

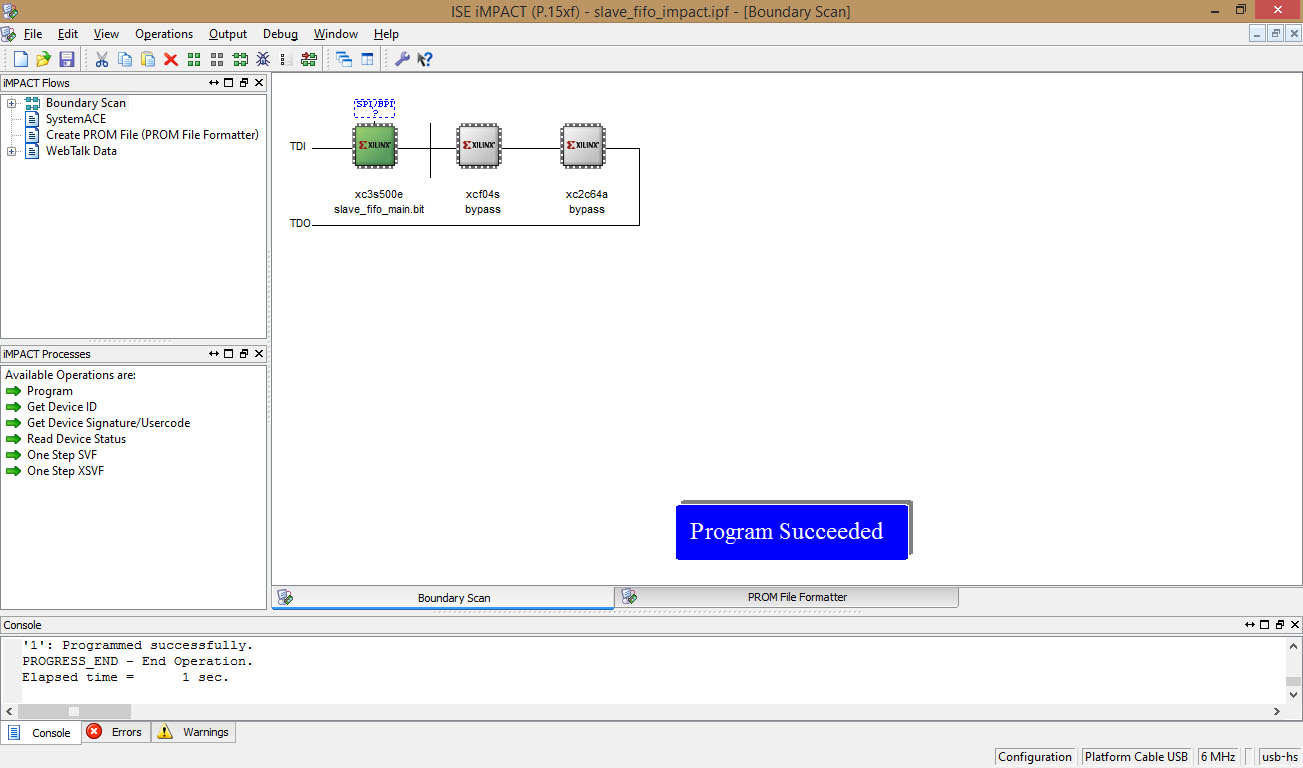
### 5.6.1. Konfiguracja z wykorzystaniem interfejsu JTAG

Pierwszym zadaniem przy wybraniu tej opcji jest synteza projektu oraz wygenerowanie pliku do zaprogramowania układu. W tym celu należy wybrać opcję *Generate Programming File* widoczną pod drzewem projektu. Jeśli wszystko przebiegło poprawnie to w konsoli powinien pojawić się komunikat *Process "Generate Programming File" completed successfully*, który informuje użytkownika o braku problemów podczas syntezy i kompilacji (Rys. 25).



Rys. . Komunikat w konsoli upewnia, że plik wygenerowano poprawnie

Po wygenerowaniu pliku wybrano opcję *Configure Target Device*. Uruchomiono program Impact ISE, w którym wykryto oraz przypisano do układu FPGA plik z konfiguracją. Po lewej stronie okna wybrano opcję *Boundary Scan*, potem w białym polu okna naciśnięto prawy przycisk myszy i wybrano opcję *Initialize Chain*. Dzięki temu wskazano plik slave\_fifo.bit, który uprzednio wygenerowano. Kolejne dwie opcje to *Bypass*, czyli pominięcie dodatkowych możliwości konfiguracji. Wybrano pierwszy układ XC3S500E oraz naciśnięto opcję *Program*. Po poprawnym zaprogramowaniu układu FPGA ukazała się informacja *Program Succeeded* (Rys. 26).



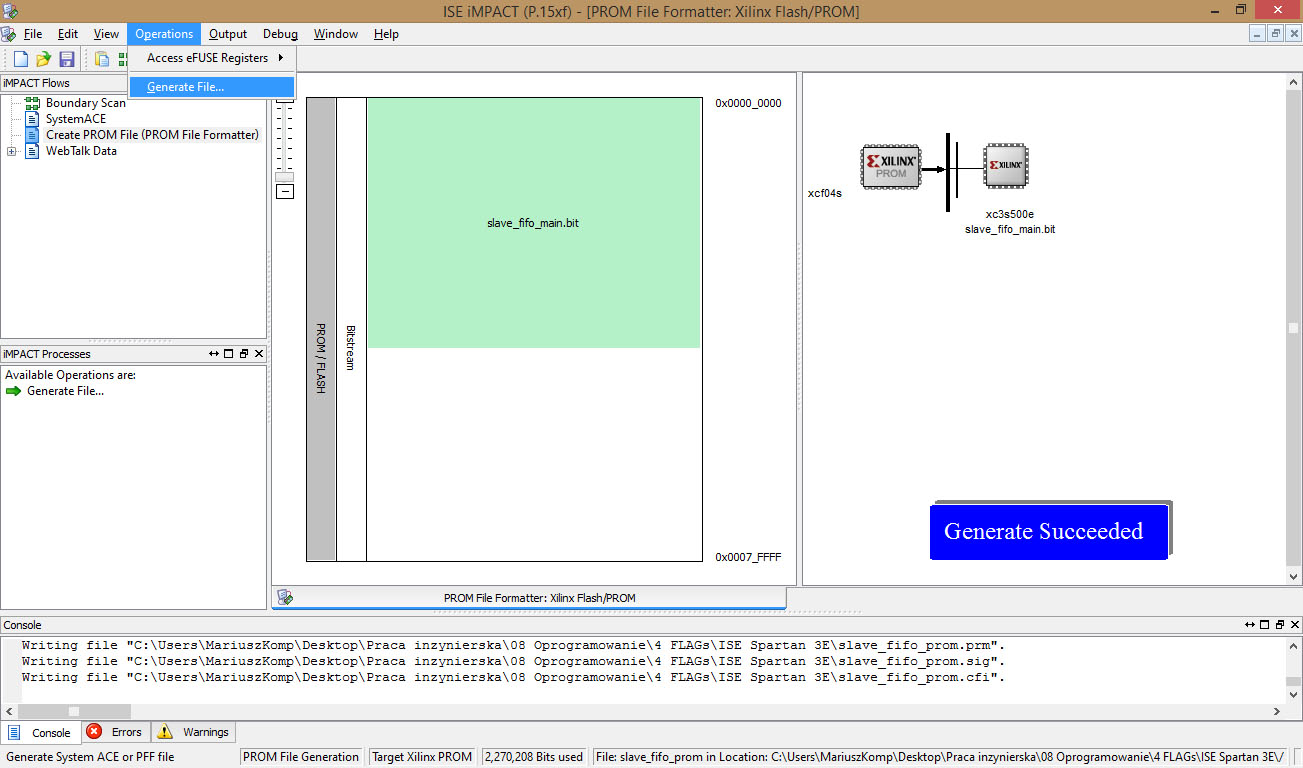
Rys. . Informacja o poprawnym zaprogramowaniu układu FPGA

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

Analogicznie jak w poprzedniej konfiguracji najpierw należy wygenerować plik .bit używając polecenia *Generate Programming File*. Po lewej stronie wybrano *Create PROM File (PROM File Formatter)*. Ukazało się kolejne okno, w którym wybrano:

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

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



Rys. . Generacja 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*. Dzięki tej konfiguracji po każdym uruchomieniu zestawu układ programowalny wykonywał program, który wcześniej przygotowano.

# 6. Przedstawienie wyników oraz efektów działania interfejsu

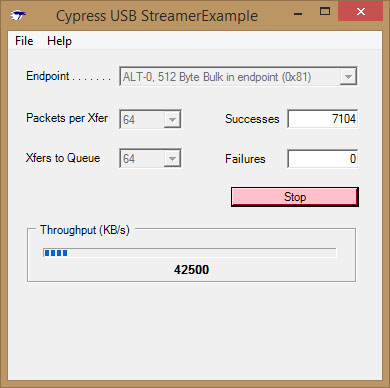
Po zaprezentowaniu teorii i praktycznej realizacji oprogramowania dla obu zestawów ewaluacyjnych przyszedł czas na przedstawienie wyników transmisji danych.

## 6.1. Ciągłe wysyłanie danych z FPGA do komputera

Po uruchomieniu i zaprogramowaniu obu układów użyto przełącznika oznaczonego na płytce Spartan 3E kodem SW2, dzięki czemu program przeszedł ze stanu bezczynności do trybu ciągłego wysyłania danych z FPGA do komputera. Można zaobserwować na ekranie LCD napis:

FSM: STREAM IN  
0000111100001111

W drugiej linii LCD umieszczono dwa bajty danych zapisanych w reprezentacji bitowej, które są aktualnie wysyłane do PC. W części teoretycznej napisano, że zdecydowano się wysyłać stały ciąg bitów, tzn. 0101011101001101 (0x574d), a nie 0000111100001111 (0x0f0f). Dzieje się tak, ponieważ nie uruchomiono programu Cypress Streamer, który odpowiada za ciągły odbiór danych w komputerze (lub ich odczyt, co będzie przedstawione w następnym podrozdziale).



Rys. . Cypress Streamer, odbieranie danych w PC wysyłanych z FPGA

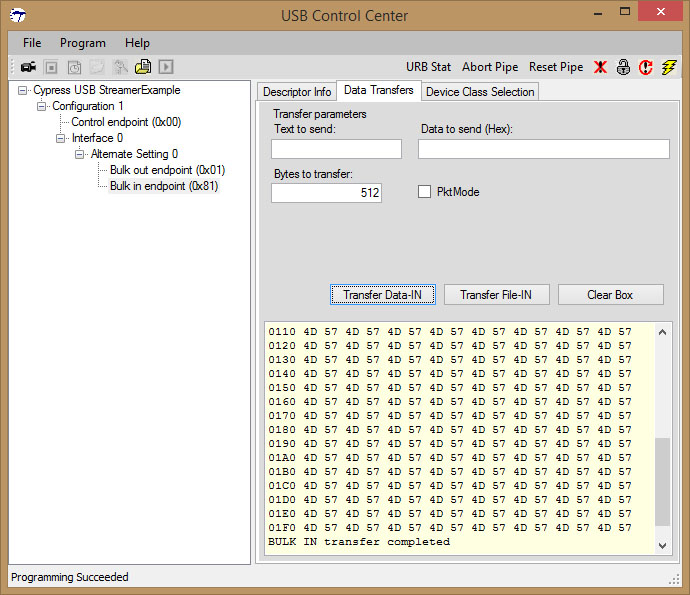
Dopiero gdy uruchomi się transmisję na komputerze (wybierając odpowiedni endpoint oraz naciskając przycisk *Start*) na ekranie LCD zmieni się napis na:

FSM: STREAM IN  
0101011101001101

Jest to potwierdzenie, że dane są wysyłane bezproblemowo. Weryfikację poprawnego odbierania danych w komputerze można przeprowadzić na dwa sposoby.

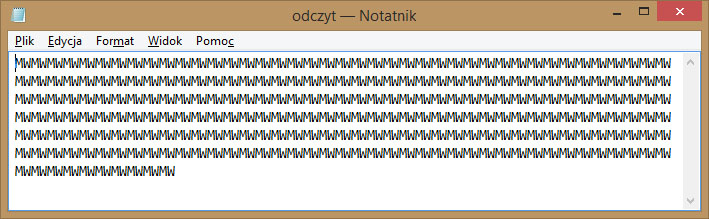
**Sposób 1**: Na Rys. 28 przedstawiono okno programu Cypress Streamer podczas pracy. Wybrano endpoint wejściowy (Bulk in, 0x81). Można zauważyć, że w sekcji *Successes* widnieje liczba 7104 (cały czas rośnie wraz z przesłanymi pakietami), natomiast w sekcji *Failures* 0 (ani jeden pakiet nie został utracony).

**Sposób 2**: Aby bezpośrednio odczytać pakiet danych w programie USB Control Center należy wybrać endpoint wejściowy z drzewa konfiguracji (Bulk in 0x81), przełączyć się na zakładkę *Data Transfers* i kliknąć lewym klawiszem myszy na przycisk *Transfer Data-IN*. W oknie głównym można zaobserwować pakiet danych zapisany heksadecymalnie (Rys. 29).



Rys. . Pobranie pakietu danych z FPGA

Opcja *Transfer File-IN* wykonuje identyczną operację, jednakże (jak sama nazwa wskazuje) dane zapisujemy do pliku tekstowego. Plik można otworzyć w dowolnym edytorze tekstowym (Rys. 30).



Rys. . Pakiet danych zapisany do pliku, jak widać 0x574d odpowiadają literom MW w kodzie ASCII

## 6.2. Ciągłe wysyłanie danych z PC do FPGA

Transmisja wygląda analogicznie jak w poprzednim przypadku. Po przejściu do trybu ciągłego odbierania danych w FPGA (wybranie przełącznika SW1) bez uruchomionego narzędzia Cypress Streamer na LCD widnieje napis:

FSM: STREAM OUT  
1111000011110000



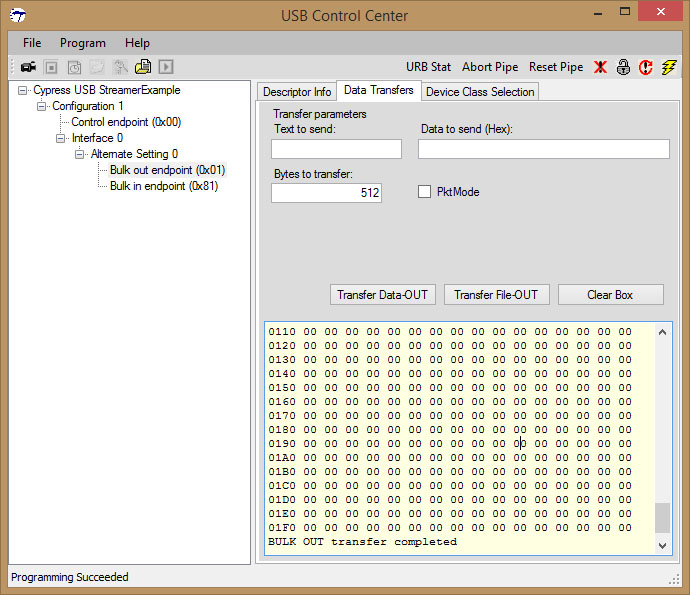
Rys. . Ciągłe wysyłanie danych z PC do FPGA

Tym razem zastosowano ciąg 0xf0f0 zamiast 0x0f0f, aby odróżnić obie transmisje. Po wybraniu endpointu wyjściowego USB (Bulk out, 0x01) i naciśnięciu przycisku *Start* (Rys. 31) zaobserwowano zmianę napisu w drugiej linii na LCD:

FSM: STREAM OUT  
0000000000000000

Jest to informacja dla użytkownika, że odbiór danych w FPGA pokrywa się z danymi wysyłanymi poprzez magistralę USB (same zera, heksadecymalnie 0x0000) z komputera.

Narzędzie USB Control Center również umożliwia wysyłanie danych. Po wybraniu odpowiedniego endpointu (Bulk out, 0x01) i naciśnięciu przycisku *Transfer Data-OUT* zanotowano w głównym oknie ciąg danych złożony z samych zer (Rys. 32).



Rys. . Wysłanie pakietu do FPGA

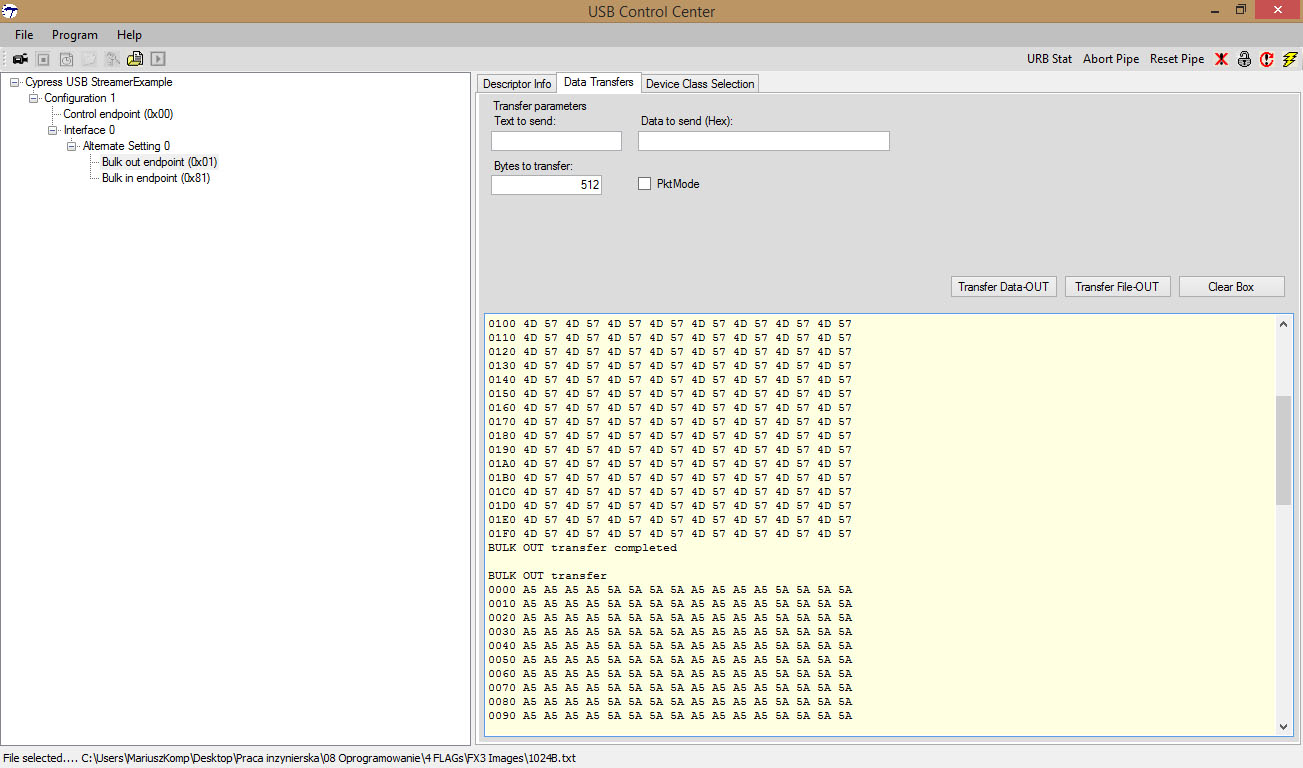
## 6.3. Pętla

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. Dwa poprzednio opisywane tryby działają używając jednego programu. Aby uruchomić tryb pętli w FX3 trzeba 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 pętli.

Na płytce Spartan 3E załączono pierwszy przełącznik oznaczony jako SW0, jest on odpowiedzialny za wejście do trybu pętli ze stanu bezczynności. Na ekranie LCD widnieje tekst:

FSM: LOOPBACK  
0000000000000000

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 endpointu wysyłkowego (Bulk out, 0x81) i wybraniu opcji Transfer File-OUT wysłano do FPGA trzy pakiety danych po 512 bajtów. Najpierw wysłano pakiet 0x574d, a potem dwa pakiety 0xa5a5. Ilustruje to Rys. 33.



Rys. . Wysłano 3 pakiety danych, jeden 0x574d, a później dwa 0xa5a5

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 ukazał się ciąg złożony z 0x574d, a dopiero potem dwa ciągi 0xa5a5. Zgadza się to z zasadą działania pamięci FIFO, ponieważ najpierw odebrano dane, które zostały wysłane jako pierwsze (ang. First In, First out).

# 8. Podsumowanie

Podsumowanie wszystkiego