Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Systemów Elektronicznych

Praca dyplomowa inżynierska

na kierunku Elektronika
w specjalności Elektronika i Fotonika

Przenośny system do diagnostyki i uruchamiania systemów cyfrowych realizowanych w układach FPGA

Paweł Murdzek
Numer albumu 310850

promotor
dr hab. inż. Wojciech Zabołotny

WARSZAWA 2025

Przenośny system do diagnostyki i uruchamiania systemów cyfrowych realizowanych w układach FPGA

Streszczenie. Przenośny system do diagnostyki i uruchamiania systemów cyfrowych realizowanych w układach FPGA powinien umożliwiać wygodne i powtarzalne debugowanie dostępnych systemów FPGA od różnych producentów - między innymi układów firm AMD oraz Intel®. Obecnie na rynku rozwiązania, które proponują producenci różnią się od siebie i nie są "kompatybiline"z FPGA konkurencji, co utrudnia jednoczesną pracę na FPGA należących do różnych ekosystemów. Potrzebne jest narzędzie o charakterze otwartym, które pozwoliłoby na ujednolicenie owego procesu. Przedstawiony w pracy system system ma ujednolicić proces, jednocześnie umożliwiając komfortowe dodawanie nowych FPGA do bazy modułów oraz jego łatwą integrację z projektami użytkownika.

Słowa kluczowe: FPGA, DEBUG, UNIWERSALNY Spis listingów

Portable system for diagnostics and start-up of digital systems implemented in FPGA systems

Abstract. A portable system for diagnostics and commissioning of digital systems implemented in FPGA systems should enable convenient and repeatable debugging of various FPGA systems from different manufacturers - among others, AMD and Intel® chips. Currently, on the market, solutions offered by those companies are independent of each other and not necessarily similar in use, which makes simultaneous work on FPGAs belonging to different ecosystems difficult. There is a lack of an open-source tool that would streamline the process. Tool presenten in this thesis aims to standardize the process, while enabling convenient addition of new FPGAs to the database of modules as well as integrating with user projects.

Keywords: FPGA, DEBUG, UNIVERSAL

załącznik nr 3 do zarządzenia
nr 28 /2016 Rektora PW

Politechnika Warszawska

……..
miejscowość i data

……..
imię i nazwisko studenta

……..
numer albumu

…….…….
kierunek studiów

OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.

.....
czytelny podpis studenta

Spis treści

1. Wprowadzenie

1.1. Cel pracy

Celem niniejszej pracy jest stworzenie uniwersalnego systemu, który pozwoli użytkownikowi na powtarzalną oraz łatwą weryfikację projektów stworzonych w języku opisu sprzętu. Weryfikacja ta będzie się odbywać w czsei rzeczwystym na badanym sprzęcie, a nie w środowisku symulacyjnym.

1.2. Zakres projektu

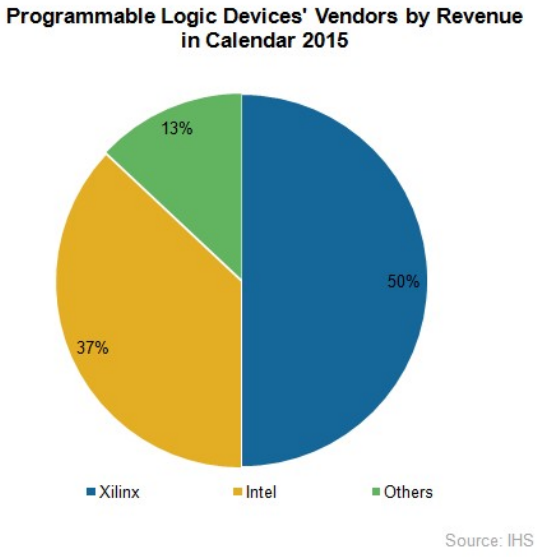
Projektowane narzędzie o charakterze open-source, będzie wykorzystywać inne narzędzia o charakterze otwartym. Za jego pomocą powinniśmy być w stanie walidować oraz uruchamiać FPGA różnych dystrybutorów, nie będąc zmuszonymi polegać na środowiskach przez nich przygotowanych. Ze względu na założenie przenośności systemu, rozwiązanie powinno być jak najbardziej uniwersalne i proste. Z racji na charakter otwarty tego projektu, warto zwrócić uwagę również na cenę rozwiązania, aby nie była ona przeszkodą w wykorzystywaniu systemu w praktyce.

1.3. Metodyka pracy

1.3.1 Analiza rynku:

a) Badanie rozwiązań udostępnionych przez obecnych i przeszłych deweloperów [1][2][3].

b) Sprawdzenie udziału poszczególnych dostawców w celu wyznaczenia priorytetu rozwoju kolejnych rozwiązań [4]



Rysunek 1.1. Udział w rynku FPGA poszczególnych producentów [5].

1.3.2 Wykonanie projektu debuggera z wstępnymi założeniami obranymi po analizie rynku opisanej w punkcie 1.a). Ograniczenie się do jednej płytki SoC na producenta.

1.3.3 Weryfikacja działania projektu na FPGA o różnej architekturze.

1.3.4 Dodanie kolejnych układów FPGA od tych samych producentów lub dodanie kolejnych producentów.

1.3.5 Ogólna rozbudowa projektu o dodatkowe metody umożliwiające wygodniejszą analizę przebiegów sygnałów w badanych modułach.

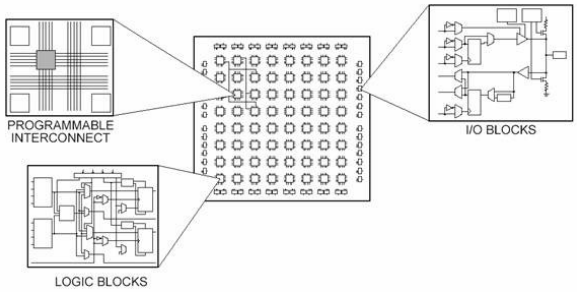
2. Układy FPGA i ich zastosowanie

2.1. Wprowadzenie do FPGA

FPGA - field programmable gate array (pl. bezpośrednio programowalna macierz bramek) to uniwersalne układy scalone złożone z bramek logicznych, które pozwalają na łatwe tworzenie prototypów rozwiązań bez konieczności zmiany samego sprzętu. W odróżnieniu od tradycyjnych ASIC - application-specific integrated circuit pozwalają na wielokrotne zmiany w samym układzie poprzez zmianę samego układu, którą dokonuje się programując daną płytkę w języku opisu sprzętu [6]. Układy FPGA zapewniają większą uniwersalność ze względu na porty GPIO, które w większości mogą być dowolnie konfigurowalne w przeciwieństwie do portów dedykowanych z małą możliwością zmiany funkcjonalności. Dodatkowo pozwalają na wykonywanie wielu zadań jednocześnie i równolegle w przeciwieństwie do przełączania się między procesami w wyniku przerwań [7].

2.2. Architektura FPGA

Układy FPGA składają się z kilkudziesięciu, a w skomplikowanych układach, do kilkunastu tysięcy rozmieszczonych matrycowo bloków logicznych *(ang. CLB - configurable logic blocks)* oraz z połączeń między tymi blokami a także portów wejścia i wyjścia, pozwalającymi na komunicję systemu z zewnętrznymi układami oraz peryferiami.



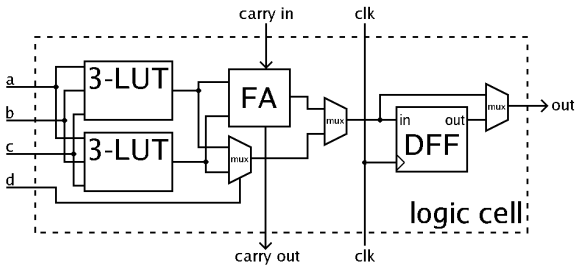
Rysunek 2.1. Ogólny schemat układu FPGA [8].

Poszczególne rozmieszczone matrycowo bloki logiczne składają się z LUT *(ang. look-up table)*, które które służą jako różnorodne bloki logiczne oraz flip-flopów które są używane jako liczniki, dzielniki oraz komórki pamięci [9].

2.3. Zastosowania FPGA

Żeby zrozumieć zastosowanie FPGA należy zwrócić uwagę na ich charakterystykę. Główne cechy układów to między innymi:

- Niskie opóźnienia pozwalające na wygodne zastosowanie do przetwarzania sygnałów w czasie rzeczywistym.



Rysunek 2.2. Przykładowa ilustracja CLB (LUT – Lookup table, FA – Full adder, DFF – D-type flip-flop) [10].

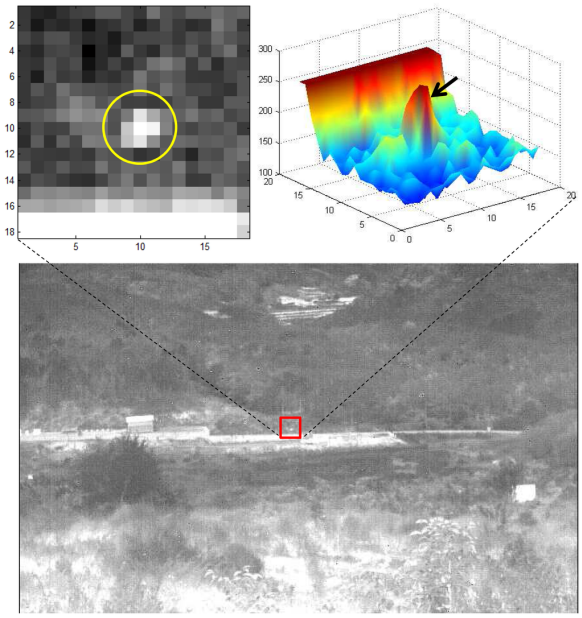
- Szybkie wprowadzanie nowych rozwiązań i modyfikacji hardware'u za pomocą zmiany w języku opisu sprzętu.
- Szerokie właściwości adaptacyjne FPGA do indywidualnych potrzeb w przeciwieństwie do tradycyjnego ASIC [7].
- Możliwość wykonywania wielu operacji równolegle podczas przetwarzania danych, nie wykorzystując przerwań [7].
- Długie "życie"układu [11].
- W określonych przypadkach potrafią zapewnić większą szybkość przetwarzania danych od tradycyjnych GPU przy jednoczesnym mniejszym zużyciu energii [11].

Wykorzystanie FPGA wiąże się jednak z długim procesem projektowym, a także wymaga specjalnych eksperckich umiejętności znajomości architektury przyrządu.

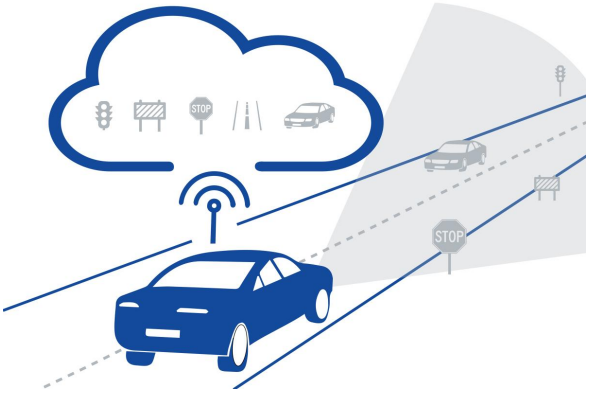
2.3.1. Klasyfikacja według przemysłu

Główne ośrodki wykorzystania FPGA w przemyśle to [11]:

1. **Przemysł kosmiczny, lotniczy oraz militarny** - ze względu na wysoką odporność na promieniowanie, znajdują one wiele zastosowań w przetwarzaniu sygnałów, obrazów [12] oraz w SDR [13].Systemy komunikacji satelitarnej oraz bezpieczne systemy komunikacyjne wymagają jednoczesnego przetwarzania dużej ilości danych, co jest dodatkowym argumentem za użyciem układów rekonfigurowalnych.
2. **Motoryzacja i mobilność** - ze względu na małe czasy opóźnienia w układach FPGA, znajdują one zastosowanie w automatycznych systemach sterowania. Dodatkowo rośnie popularność stosowania FPGA w systemach zarządzania akumulatorem samochodu [14].
3. **Robotyka i medycyna** - równoległe przetwarzanie wielu danych, pozwala na dokładniejsze zarządzanie procesami, takimi jak na przykład przetwarzanie obrazów w badaniach USG lub w obliczaniu optymalnej rotacji osi pracującej na produkcji mechanicznej ręki.
4. **Komunikacja** - ze względu na małe opóźnienia oraz możliwość implementacji umożliwiającej dużą przepustowość danych FPGA są regularnie wykorzystywane w przemyśle telekomunikacyjnym Firmy takie jak Google, Microsoft oraz Amazon wykorzystują



Rysunek 2.3. Wykorzystanie FPGA w wykrywania małych obiektów za pomocą czujników podczerwonych [12].



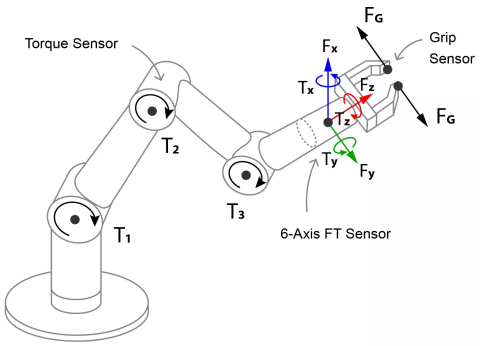
Rysunek 2.4. Jadący samochód zbiera dużą ilość danych, które muszą być szybko i trafnie przetworzone [15].

układy rekonfigurowalne w swoich serwerach, w celu przyspieszenia przetwarzanie danych.

2.3.2. Najpopularniejsze sposoby przechowywania danych konfiguracyjnych

Raport Markets.us [16], wymienia trzy stosowane technologie:

- **SRAM** - najczęściej używany sposób konfiguracji FPGA, ze względu na najbardziej uniwersalny oraz reprogramowalny charakter. Znajduje zastosowania między innymi w przemyśle motoryzacyjnym oraz centrach danych.
- **Flash** - z racji mniejszego zużycia energii, znajdują najwięcej zastosowań w elektronice użytkowej korzystającej z baterii jako główne źródło zasilania. Są to między innymi telefony komórkowe.



Rysunek 2.5. Uproszczony schemat sił działających na rękę robota podczas pracy wymagające synchronicznych wielowektorowych obliczeń.

- **Antifuse** - ze względu na tylko pojedyncza możliwość zaprogramowania, dłuższy czas życia i bezpieczeństwo wynikające z braku możliwości rekonfiguracji, znajdują najwięcej zastosowań w przemyśle kosmicznym, lotniczym oraz militarnym.

2.4. Typowe narzędzia do tworzenia oprogramowania dla FPGA

Ze względu na różnicę w architekturze FPGA, poszczególni producenci dostarczają własne oprogramowanie, służące do optymalnego zaplanowania rozstawienia wykorzystanych CLB oraz połączeń między nimi w ich układach rekonfigurowalnych. Jest to ważne dla jak najoptymalniejszego wykorzystania zasobów FPGA *(LUT oraz FF)*, jednocześnie nie wprowadzając przekłamań w zaplanowanych zadaniach oraz w przetwarzanych danych. W środowisku FPGA występuje wyścig, w którym producenci starają się przyzwyczaić konsumenta do swojego oprogramowania, a także FPGA, zapewniając układy przystosowane do różnych zastosowań. Co za tym idzie, każdy stara się zamknąć projektantów, korzystających z układów rekonfigurowalnych w swoim ekosystemie i zniechęcić do korzystania z produktów konkurencji. Przykładowe programy służące do syntezy, implementacji i generacji strumienia bitów dla FPGA:

- **Vivado Design Suite** - System firmy AMD zoptymalizowany dla skomplikowanych układów z serii 7, Ultrascale, Ultrascale+, Zynq-7000, a także nowości - Versal ACAP [1], przystosowanej pod rozwój AI we współpracy z oprogramowaniem **Vitis**.
- **Quartus® Prime** - Narzędzie firmy Intel® służące do pracy z FPGA z serii Agilex, Stratix, Arria, oraz Cyclone. Współpracuje ono z wieloma innymi narzędziami tej firmy, służącymi do niszowych zastosowań takich jak obliczenie zużycia prądu za pomocą **Intel® FPGA Power and Thermal Calculator** [17].
- **Lattice Diamond** - Obsługuje większość serii FPGA, opracowanych przez firmę Lattice Semiconductors, w tym MachXO, MachXO2, ECP2, ECP3, ECP5, CrossLink i ispMACH 4000. Pracując z FPGA z serii iCE40, iCE40 UltraPlus oraz CrossLink-NX

użyjemy innego oprogramowania, zoptymalizowanego dla projektowania z uwzględnieniem minimalizacji zużycia energii [18].

- **Gowin EDA** - Narzędzie firmy Gowin Semiconductor. Umożliwia projektowanie modułów i ich implementację na FPGA rodzin Arora V, GW1NZ oraz LittleBee [19]. Niektóre z nich charakteryzują się znacząco niższymi kosztami w stosunku do konkurencji. Ich mniejsza cena jest związana z mniejszą ilością zasobów dostępnych na FPGA.

3. Diagnostyka systemów cyfrowych

3.1. Pojęcie diagnostyki w systemach cyfrowych

SJP określa pojęcie diagnostyki *jako określanie stanu technicznego urządzeń i ustalanie źródeł awarii*[20]. W przypadku FPGA oznacza to weryfikację i walidację poprawności działania opisanego na płytce rekonfigurowalnej procesu. Do konfiguracji FPGA używamy jednego z języków opisu sprzętu takich jak Verilog lub VHDL. Weryfikację poprawności działania modułów możemy przeprowadzić przez symulację przebiegów sygnałów lub obserwując te przebiegi w czasie rzeczywistym bezpośrednio na badanej FPGA. Wykorzystywane są do tego specjalistyczne narzędzia, pozwalające na sprecyzowanie interesujących nas warunków wyzwolenia, po wywołaniu których następuje rejestracja przebiegów do pamięci. Wykrywanie błędów w czasie projektowania modułów, a także po syntezie oraz implementacji jest nieodłączną częścią programowania. Nazywamy je potocznie **debugowaniem**.

3.2. Popularne systemy diagnostyki systemów cyfrowych

3.2.1. Narzędzia przygotowane przez producentów

Każdy producent oprócz narzędzi do programowania udostępnia również narzędzia do diagnostyki czy też debugowania projektów. AMD, Intel® oraz Lattice zapewniają je jako rozszerzenie do narzędzia do projektowania układów FPGA, czyli odpowiednio:

1. **Chipscope** - to narzędzie, zastrzeżone przez AMD służące do debugowania sygnałów bezpośrednio na układzie rekonfigurowalnym [21]. W nowszych rodzinach FPGA zostało ono zastąpione przez **Vivado Integrated Logic Analyzer**. Jest ono zintegrowane ze środowiskiem **Vivado**, umożliwiajac projektowanie, implementację oraz weryfikację w jednej aplikacji.
2. **Signal Tap Logic Analyzer** - to odpowiednik Chipscope, należący oraz utrzymywany przez firmę Intel®. Służy do diagnostyki FPGA obsługiwanych przez nich [2]. Zintegrowany z oprogramowaniem **Quartus® Prime** [17].
3. **Reveal Analyzer** - to rozwiązanie zintegrowane z platformą **Lattice Diamond** [18] firmy Lattice Semiconductor. Pozwala na debugowanie w czasie rzeczywistym większości płytek tego producenta [22].
4. Gowin Analysis Oscilloscope - to narzędzie udostępnione przez firmę Gowin Semiconductor. Pozwala na badanie układów w czasie rzeczywistym. Zintegrowane ze środowiskiem **Gowin EDA** [23].

Dodatkowo należy zwrócić uwagę na narzędzia o charakterze otwartym, z licenją umożliwiająca użytkownikom pracę nad nimi. Takie narzędzia ze swojej natury, są mniej zależne od firmy oraz jej ekosystemu. Przy odpowiedniej implementacji, pozwalają one zastąpić oprogramowanie przygotowane przez producentów Narzędzia te możemy po-

dzielić na hardware, który wspomaga czytanie sygnałów na FPGA oraz software, który z nim współpracuje.

3.2.2. Narzędzia programistyczne

1. **OpenOCD** - Open On-Chip Debugger to uniwersalne narzędzie służące do nawiązywania połączenia z procesorami za pomocą interfacejsów **JTAG** oraz **SWD**. Pozwala między innymi programować, w ograniczonym zakresie, wymazywać pamięć oraz debugować układ scalony. W przypadku FPGA najważniejszym zaimplementowanym zastosowaniem jest utworzenie mostku komunikacyjnego z układem rekonfigurowalnym oraz wysłanie komend za pomocą wbudowanego w to narzędzie interpretera TCL. Narzędzie pozwala również na wgrywanie ciągu bitowego na FPGA [24].
2. **Sigrok** - to pakiet o charakterze otwartego kodu źródłowego, wspomagający analizę sygnałów. Obsługuje analizatory logiczne, oscyloskopy i multimetry, z których rejestruje sygnały i pomaga w ich analizie. W celu przeprowadzenia analizy graficznej wykorzystuje PulseView. Zapewnia dekodowanie popularnych protokołów komunikacyjnych takich jak I2C, SPI, UART. Może posłużyć do przechwytywania i analizowania sygnałów FPGA w celu debugowania [25].
3. **GTKWave** - to narzędzie pozwalające na wizualną analizę działań napisanych w języku opisu sprzętu modułów na podstawie przebiegów sygnałów zapisanych w plikach o formatach [26]:
   - `.VCD`: Value Change Dump
   - `.FST`: Fast Signal Trace
   - `.LXT`: InterLaced eXtensible Trace
   - `.LXT2`: InterLaced eXtensible Trace Version 2
   - `.VZT`: Verilog Zipped Trace
   - `.IDX`: VCD Recorder Index File
   - `.GHW`: GHDL Wave File
   - `.AET2`: All Events Trace Version 2
   - `.VPD`: VCD Plus Dump
   - `.WLF`: Wave Log File
   - `.FSDB`: Fast Signal Database

   Jest to narzędzie szczególnie przydatne do debugowania problemów z wynikających z zależności czasowych oraz logiki współpracujących ze sobą modułów przez symulację.
4. **Analizatory protokołów komunikacyjnych (USBlyzer/Wireshark)** - to narzędzia do debugowania protokołów komunikacyjnych, które mogą być zaimplementowane między innymi w projektach FPGA. **USBlyzer** [27] jest przeznaczony do weryfikacji poprawności przesyłanych danych za pomocą interfejsu USB, natomiast **Wireshark** [28] służy przechwytywania danych wysyłanych w sieci Ethernet.

5. **fpgadbg2** - to narzędzie do rejestrowania próbek sygnałów FPGA. Po określonym przez projektanta wyzwoleniu, dane wejściowe są zapisywane w pamięci FPGA, a następnie są przesyłane do komputera za pomocą interfejsu UART. Po przetworzeniu są wyświetlane za pomocą aplikacji GTKWave [3].
6. **LiteScope** - to zintegrowany w układach scalonych analizator logiki w projektach LiteX SoC[29]. Zapewnia monitorowanie w czasie rzeczywistym wewnętrznych sygnałów FPGA bez konieczności stosowania dodatkowego zewnętrznego sprzętu oraz oprogramowania [30]. Posiada wbudowane narzędzie wishbone-tool, pozwalające na monitorowanie ruchu na magistrali wishbone.
7. **LiteETH** - to część pakietu LiteX, zapewnia narzędzia do debugowania komunikacji interfejsu Ethernet w projektach SoC wykorzystujących FPGA z tej rodziny. Obejmuje symulację i obsługę debugowania w czasie rzeczywistym [31].
8. **OpenTitan Debug Infrastructure** - to środowisko do weryfikacji dla projektów stworzonych w środowisku OpenTitan. Oferuje bezpieczny dostęp do debugowania, integrację JTAG i obsługę **STM32 NUCLEO-L552ZE-Q** jako debuggera, nazywanego przez deweloperów OpenTitan **HyperDebug Board** [32].

3.2.3. Narzędzia sprzętowe

1. **Openbench Logic Sniffer** - to narzędzie służące do przechwytywania i zapisywania sygnałów z elektrycznych. Może służyć do debugowania FPGA i innych układów scalonych. Jest wyposażony w piny i sondy. Ze względu na swoją dużą szybkość próbkowania, jest skuteczny w obserwacji pinów FPGA. Do pełnej analizy potrzebuje programu typu Sigrok lub PulseView [33].
2. **Mostki komunikacyjne** - to istnieje wiele narzędzi pozwalających na stworzenie interfejsu komunikacyjnego pomiędzy badanym modułem a stacją badawczą. Takie konwertery sygnałów to może być między innymi:
   - konwerter USB-UART,
   - konwerter USB-SPI,
   - konwerter USB-I2C.
3. **RaspberryPI** - to komputer jednopłytowy, który zawiera kontrolery typowych niskopoziomowych interfejsów komunikacyjnych (np. I2C, SPI) oraz pozwala na wydajną programową realizację interfejsu JTAG bez konieczności stosowania dodatkowych mostków. Odkryte na PCB piny, umożliwiają bezpośrednie połączenie z peryferiami. Przy niektórych interfejsach, konieczne może się okazać wykorzystanie konwertera poziomów logicznych. Dodatkowo niektóre modele z rodziny RaspberryPI mają możliwość instalacji RaspberryPI OS - zmodyfikowanego jądra linux na bazie systemu Debian. System ten może nam posłużyć do instalacji programów wspierających analizę takich jak GTKWave lub Verilator. Modele RaspberryPI umożliwiające instalację systemu RaspberryPI OS:

- Raspberry Pi 5,
- Raspberry Pi 4 (Model B),
- Raspberry Pi 400,
- seria Raspberry Pi 3 *(Modele B, B+, A+)*,
- seria Raspberry Pi Zero *(modele Zero, Zero W, Zero 2 W)*,
- Raspberry Pi 2,
- Raspberry Pi 1
- Raspberry Pi Compute Module 1,
- seria Raspberry Pi Compute Module.

3.2.4. Narzędzia do debugowania poprzez symulację układu

Przed weryfikacją działania projektu w systemie warto przeanalizować logikę projektowanych modułów za pomocą symulacji behawioralnej. Producenci tacy jak:

- AMD,
- Intel®,
- Lattice Semiconductor,

integrują narzędzia pozwalające na taką analizę. Te narzędzia to:

- **Vivado Design Suite** wykorzystuje narzędzie **Xylinx Simulator** - w skrócie **XSIM**. Pozwala na symulację języków opisu sprzętu Verilog, SystemVerilog, SystemC i VHDL. Jest ono utrzymywane przez firmę Siemens AG.
- **Quartus® Prime** - wykorzystuje narzędzie **ModelSim**, przystosowane do pracy z FPGA firmy Intel®. Narzędzie to jest utrzymywane przez firmę Siemens AG.
  Lattice Diamond - wykorzystuje narzędzia **ModelSim** oraz **Active-HDL**, utrzymywane przez firmę Aldec.

Na rynku istnieją również rozwiązania niezależne, są nimi między innymi:

1. **QuestaSim** - to narzędzie utrzymywane przez firmę Siemens AG. W odróżnieniu od **ModelSim**, służy do symulacji projektów, złożonych z wielu milionów bramek. Mogą być one opisane w językach SystemVerilog i VHDL [34].
2. **Incisive® Enterprise Simulator** - to rozwiązanie, które wspiera symulację modułów opisanych w językach *e, Verilog, VHDL, SystemC, SystemVerilog, CPF, PSL, OVL, SVA, Simulation* [35]. Stworzone i utrzymywane przez firmę Cadence.
3. **VCS** - to narzędzie utrzymywane przez firmę Synopsis. Umożliwia symulację języków Verilog, VHDL i SystemVerilog 3.1a [36].
4. **Cocotb** - to biblioteka Pythona służąca do pisania środowisk testowych opartych na współpracy z zintegrowanymi programami wykorzystywanymi w projektach FPGA i ASIC. Umożliwia testowanie protokołów, kosymulację za pomocą symulatorów, takich jak ModelSim, Icarus Verilog i Verilator oraz weryfikację logiki FPGA [37].
5. **Verilator** - to symulator i kompilator Verilog o otwartym kodzie źródłowym. Konwertuje język Verilog na C++ w celu symulacji o dużej szybkości. Służy do analizy czasu

z dokładnością do cyklu i rozległego testowania w symulacjach FPGA. Pozwala na weryfikację modułów napisanych w języku sprzętu na bazie symulacji [38].

6. **SymbiFlow** - to zbiór narzędzi FPGA, o otwartym kodzie źródłowym obsługujący wiele rodzin układów FPGA. Integruje narzędzia o otwartym kodzie źródłowym oraz te zamknięte na ingerencję użytkowników z zewnątrz. Do debugowania używa narzędzia otwartego - **VPR**, które pozwala na badanie schematu i zastawianie pułapek w weryfikowanym schemacie architetury. Całość działa na bazie symulacji, a nie na fizycznym układzie. [39][40].
7. **IceStorm** - to część łańcucha narzędzi o otwartym kodzie źródłowym dla układów FPGA obsługiwanych przez YosysHQ. Jego główne zastosowanie to dokumentacja oraz tworzenie strumienia bitowego dla układów rekonfigurowalnych Lattice iCE40. Pozwala on dokonywać inspekcji zasobów oraz zapewnia narzędzie do debugowania zależności czasowych na podstawie pliku strumienia bitów, wykorzystując zintegrowane narzędzie *icetime* [41].

3.3. Problemy z popularnymi systemami diagnostyki FPGA

Biorąc pod uwagę wszystkie narzędzia dostępne na rynku, nie ma jednego rozwiązania o naturze otwartej, które pozwalałoby na proste i uniwersalne debugowanie w czasie rzeczywistym FPGA o różniącej się od siebie architekturze. W przypadku uruchamiania układów stosujących różne, wyspecjalizowane FPGA, takie uniwersalne narzędzie ułatwiłoby prace inżynierowi szukającemu błędu na danym układzie. Dodatkowo takie narzędzie nie powinno być trudne do zaimplementowania, a co za tym idzie utrudniać kompilacji. Nie powinno wprowadzać przekłamań w działaniu opisywanych modułów po implementacji a także nie powinno zużywać wielu zasobów FPGA. Korzystanie z narzędzi dostępnych przez producentów, wymusza wybór między korzystaniem z pełni funkcjonalności FPGA lub debugowaniem systemu. Jednym z problemów może być to, że wybrane przez nas SoC nie wspiera dwóch połączeń JTAG za pomocą jednego złącza, co uniemożliwia komunikację z PC, aby wysyłać dane przez JTAG, oraz jednoczesny podgląd sygnałów. Jeśli nasz system wymaga przesyłania danych przez JTAG, proces debugowania może być utrudniony. Narzędzia dostępne na rynku, które cechuje przenośność są kosztowne, trudno dostępne lub ograniczone w swoich możliwościach. Natomiast te bardziej zaawansowane, są zamknięte w swoich ekosystemach a ich przenośność jest ograniczona do jednego producenta. Korzystanie z wielu z nich jednocześnie może być niewygodne. Tym problemom ma za zadanie wyjść na przeciw **Przenośny system do diagnostyki i uruchamiania systemów cyfrowych realizowanych w układach FPGA**, zapewniając użytkownikom wygodny, łatwy w obsłudze oraz uniwersalny system do debugowania układów FPGA, jednocześnie ograniczając jego koszty do minimum.

4. Projektowanie przenośnego systemu diagnostycznego

4.1. Wymagania projektowe

Przenośny system do diagnostyki i uruchamiania systemów cyfrowych realizowanych w układach FPGA musi spełniać podstawowe założenia:

1. Być **przenośny**, czyli działać na możliwie jak największej ilości platform FPGA, wyprodukowanych przez różnych producentów, posiadające różne architektury oraz inne środowiska do debugowania projektów.
2. Umożliwiać **diagnostykę** oraz **uruchomienie** systemów cyfrowych po syntezie oraz implementacji. W niektórych projektach, podczas uruchomienia, pojawiają się błędy, których nie było widać podczas symulacji układu ze względu na oddziaływanie zewnętrznych czynników, potencjalną długość ścieżek, lub zakłócenia powodowane przez zewnętrzne układy elektroniczne. Ten system ma umożliwić namierzenie oraz wyeliminowanie takich błędów.

Dodatkowo przy projekcie takiego systemu należy uwzględnić jego użyteczność i wygodę dla końcowego użytkownika. Biorąc te fakty pod uwagę, dodatkowymi założeniami tego projektu są :

1. Komfort przenoszenia - system ten powinien być użytkowo taki sam, lub znacznie zbliżony dla każdego rodzaju FPGA, możliwym do debugowania. Możliwość rozwoju, w tym dodawanie nowych FPGA, modyfikowanie połączeń oraz wykorzystanych pinów powinno być stosukowo proste, tak aby móc łatwo było przenosić system między projektami.
2. Dostępność - materiały, czyli oprogramowanie, wraz ze sprzętem potrzebnym do uruchomienia systemu diagnostycznego powinny być łatwo dostępne oraz dobrze udokumentowane.

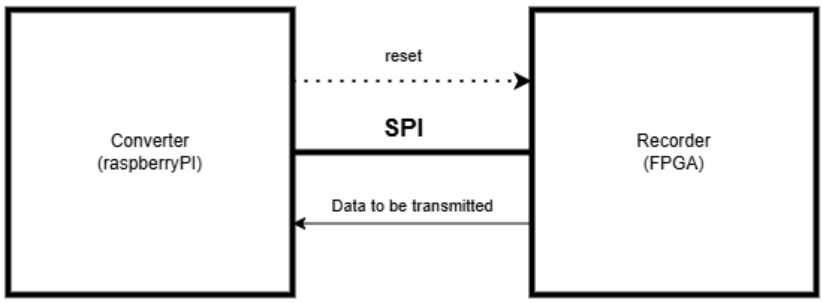
4.2. Wyboru platformy sprzętowej oraz zastosowane technologie

Ze względu na powyższe założenia zdecydowałem się rozwinąć istniejące już rozwiązanie fpgadbg2 [3]. Językiem opisu sprzętu, na który się zdecydowałem to VHDL. Aby projekt stał się łatwo dostępny postanowiłem użyć dodatkowo platformy Raspberry Pi[42], która służy w tym projekcie jako analizator nagranych sygnałów oraz osobisty komputer do prezentacji przebiegów na FPGA. Konwerter sygnału jest stworzony w języku Python, z użyciem bibliotek w języku C udostępnionych przez GTKWave, służących do stworzenia pliku *.lxt*, umożliwiającego wyświetlanie próbek w przeglądarce GTKWave [43].

4.3. Opis projektu systemu

Projekt składa się z trzech głównych części, a dokładniej z:

- Rejestratora sygnałów, zamiennie nazywanego w dalszej części **fpgadbg core**. Jego zadaniem jest zapisanie określonej ilości próbek w pamięci FPGA po wydarzeniu zdefiniowanym przez użytkownika.
- Analizatora danych, zamiennie nazywanego w dalszej części **fpgadbg conv**. Jego zadaniem jest wysłanie bajtów konfiguracyjnych do FPGA, a następnie odebranie od niej sygnału z danymi. Po zapisaniu ich oraz obróbce, powinnybyć je przetworzone do pliku w formacie *.lxt*. Na koniec powinien on wykonać prezentację danych za pomocą GTKWave.
- Interfejsu komunikacyjnego, zamiennie nazywanego w dalszej części **wrapperem**. Jego zadaniem jest zapewnienie połączenie pomiędzy uruchamianym projektem oraz zewnętrznym konwerterem. Ze względu na możliwości sprzętowe RaspberryPI, zrobione zostaną bezpośrednie połącznia między pinami RaspberryPI i badanego SoC, zawierającego FPGA za pomocą przewodów.



Rysunek 4.1. Uproszczony schemat projektu przenośnego system do diagnostyki i uruchamiania systemów cyfrowych realizowanych w układach FPGA.

5. Implementacja systemu

5.1. Komponenty składowe

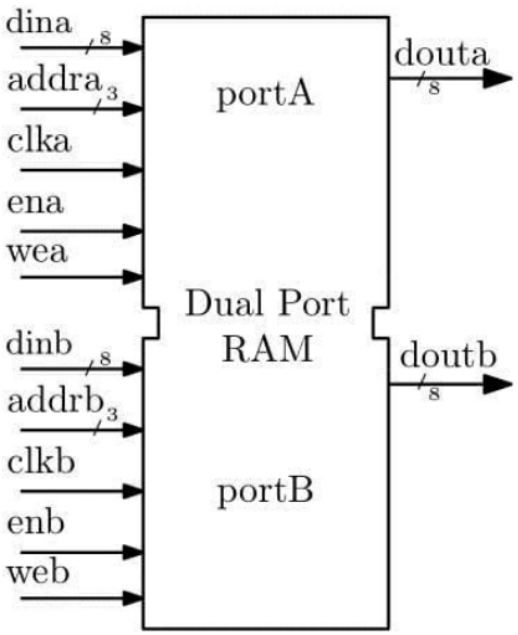
5.1.1. Rejestrator sygnałów

Pierwszą częścią projektu jest moduł pozwalający na obserwację w czasie rzeczywistym sygnałów na FPGA oraz ich zapis do pamięci. Pamięć, w której zapisane próbki mają być przechowywane powinna być łatwo dostępna na każdej FPGA, w której działa **Przenośny system do diagnostyki i uruchamiania systemów cyfrowych realizowanych w układach FPGA**. Nie powinny one wymagać sekwencyjnej inicjalizacji, żeby umożliwić narzędziu rejestrację sygnałów w każdej, także początkowej, fazie działania walidowanego projektu. System ten nie powinien opierać się na zewnętrznych peryferiach, aby ułatwić jego integrację do różnych układów wykorzystujących FPGA. Układy rekonfigurowalne implementują bloki pamięci BRAM, w dokumentacjach technicznych nazywane zamiennie *EBR* [44], które mogą zostać wykorzystane jako pamięć do zapisania próbek. W zależności od architektury FPGA bloki te mają typowo wielkość:

- 18Kb lub 36Kb dla układów rekonfigurowalnych wykonanych w architekturze wykorzystywanej przez AMD, a wcześniej Xylinx;
- 640b, 9Kb, 20Kb, 144Kb dla FPGA stworzonych w architekturze wykorzystanej przez Intel®.

Implementacja bloków pamięci BRAM w układach FPGA za pomocą języków opisu sprzętu jest dokładnie udokumentowanym zagadnieniem. W sieci istnieje wiele rozwiązań implementujących różne typy bloków BRAM, różniących się od siebie zajętością zasobów oraz możliwymi do przeprowadzenia operacjami. Wiele z nich jest udostępnionych na zasadzie otwartej licencji. W budowanym narzędziu, konieczna jest implementacja DPRAM, które umożliwi zapis próbkowanych sygnałów do pamięci w czasie rzeczywistym podczas badania sygnału w określonym momencie wyzwolenia oraz późniejszy odczyt w celu przesłania danych do urządzenia peryferyjnego, aby umożliwić ich dalszą analizę. W budowie rejestratora sygnałów wykorzystano implementację, stworzoną w narzędziu fpgadbg2 [3] ze względu na zintegrowany z nią moduł wysyłający do zewnętrznych interfejsów podstawowe dane o zainstancjonowanej pamięci. Po drobnych zmianach, jako dodatkowy sposób weryfikacji poprawności pracy z narzędziem, przesyła ona także informację zwrotną dla użytkownika o liczbie deklarowanych sygnałów. Moduł ten implementuje pamięć TDPRAM, która pozwala na jednoczesny i odczyt danych.

Istnieje możliwość modyfikacji objętości wykorzystanych zasobów, zwiększając lub zmniejszając liczbę próbek, które są domyślne rejestrowane, jak i również długość badanego sygnału wejściowego, którego próbki będą zapisywane. Wiąże się to z modyfikacją deklaracji wyżej wymienionych informacji przed syntezą narzędzia, a co za tym idzie modyfikacją liczby zainstancjonowanych komórek pamięci. Rozmieszczenie wykorzystanych



Rysunek 5.1. Schemat komórki TDPRAM [45].

zasobów na FPGA, zarządzane przez narzędzia wykorzystywane w procesie, odbywa się podczas implementacji projektu na FPGA.

Ta część projektu została wykonana w języku opisu sprzętu VHDL; jest ona implementowana bezpośrednio w FPGA. Ze względu na różnice w FPGA różnych dystrybutorów, wynikające z architektury oraz wymagań tekstowych ich kompilatorów jak i również różnice między FPGA tego samego dystrybutora, wynikające z innego wyprowadzenia pinów oraz zastosowania innego oscylatora, potrzebna będzie modyfikacja modułu głównego systemu do diagnostyki i uruchamiania systemów cyfrowych. Zostały przygotowane przykładowe projekty, które mogą pomóc w lepszym zrozumieniu i implementacji narzędzia.

5.1.2. Analizator danych

Aby umożliwić walidację działania badanego projektu zaimplementowanego na FPGA, należy pobrać zapisane w FPGA dane oraz zaprezentować je użytkownikowi. Do stworzenia modułu to umożliwia wykorzystano RaspberryPI 5. System operacyjny RaspberryPI OS posiada wsparcie dla większości standardowych programów, bibliotek oraz sterowników dostępnych dla systemu Debian, na którym jest oparty. Odkryte piny sprzętu pozwalają na bezpośrednie połączenie z debugowanym układem rekonfigurowalnym. Do komunikacji między modułami RaspberryPI oraz FPGA wykorzystano interfejs SPI. Sterownik, przystosowany do pracy z **rejestratorem sygnałów** napisano w języku Python, wykorzystując bibliotekę SPIDEV [46]. Prezentacja zapisanych w pamięci BRAM danych w postaci "waveform"jest wykonana za pomocą aplikacji GTKWave. Tony Bybell udostępnia, na zasadzie otwartej licencji, bibliotekę `lxt_write` [43], która pozwala na storzenie pliku w formacie `.lxt`, obsługiwanego przez ten program. Wykorzystanie tej biblioteki oraz

potrzebne operacje analizy ciągu bitów odebranych od układu rekonfigurowalnego przez RaspberryPI przed konwersją zostały wykonane w języku Python.

Jednopłytowy komputer jest odpowiedzialny za:

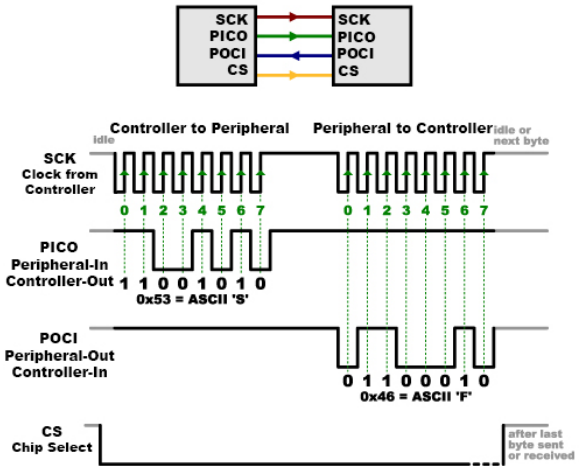
1. Skonfigurowanie Rejestratora sygnałów.
2. Wybranie badanej FPGA za pomocą linii CS.
3. Odebranie danych z zapisem przebiegów, wysłanych przez FPGA.
4. Przygotowanie danych do konwersji.
5. Sprawdzenie poprawności odebranych danych.
6. Skonwertowanie danych do formatu *.lxt*.
7. Wyświetlenie danych w GTKWave.

Aby przygotować biblioteki oraz zainstalować wymagane oprogramowanie, został przygotowany plik wykonywalny *install.sh*. Należy nadać mu uprawnienia do wykonywania, a następnie go uruchomić. Zajętość systemowa po instalacji wynosi 7,5GB. SPI wewnątrz konwertera danych jest przystosowane do działania z Rejestratorem sygnałów. *W przypadku wykorzystania w innych projektach, należy zweryfikować poprawność odbioru danych.* Testy zostały wykonane na RaspberryPI 5, jednak ze względu na wsteczną kompatybilność zastosowanych bibliotek, użycie każdego produktu z rodziny Raspberry powinno być możliwe bez modyfikacji narzędzia. Pozwala to zredukować cenę początkową Przenośnego systemu do diagnostyki i uruchamiania systemów cyfrowych realizowanych w układach FPGA.

W przypadku braku możliwości podłączenia ekranu dla wyświetlania wyników, istnieje możliwość pobrania pliku za pomocą protokołu SSH a następnie jego prezentacja na wybranym przez użytkownika komputerze. Plik z ostatniej walidacji układu rekonfigurowalnego nie jest usuwany. Wykorzystanie innego sprzętu niż RaspberryPI (np. komputer z zainstalowanym sytemem na bazie Debian OS razem z mostkiem komunikacyjnym USB-SPI) jest możliwe, ze względu na wykorzystane w projekcie biblioteki standardowe jako sterowniki interfejsów komunikacyjnych. Może to jednak wymagać modyfikacji narzędzia.

5.1.3. Opis protokołu komunikacyjnego

Wykorzystanym protokołem komunikacyjnym między RaspberrryPI oraz częścią nagrywającą próbki na FPGA jest SPI. Jest to interfejs cechujący się większą stabilnością od interfejsów niesynchronicznych takich jak UART oraz ze względu na zastosowanie większej ilości linii komunikacyjnych, mniejszymi przekłamaniami między różnymi układami bez wspólnej przestrzeni masy.



Rysunek 5.2. Schemat przebiegu standardowej komunikacji SPI [47].

W przypadku **Przenośnego systemu do diagnostyki i uruchamiania systemów cyfrowych realizowanych w układach FPGA** RaspberryPI jest sterownikiem, a FPGA urządzeniem podrzędnym. Ze względu na ograniczenia sprzętowe RaspberryPI, teoretyczna maksymalna częstotliwość pracy interfejsu to 125 MHz, jednak zaleca się ograniczenie jej do:

- 50 MHz dla krótkich i dobrych połączeń,
- mniej niż 25 MHz dla długich przewodów.

RaspberryPI pozwala na połączenie dwóch urządzeń podrzędnych do jednego interfejsu SPI, a co za tym idzie, nasz układ pozwala na debugowanie dwóch FPGA jednocześnie. *Domyślnie używane są SPI0 CS0. Możliwe jest korzystanie z innych linii komunikacyjnych, jednak potrzebne będą niewielkie przeróbki w pliku* `main_fpgadbg_SPI.py`

Deklaracja komunikacji SPI w kodzie:

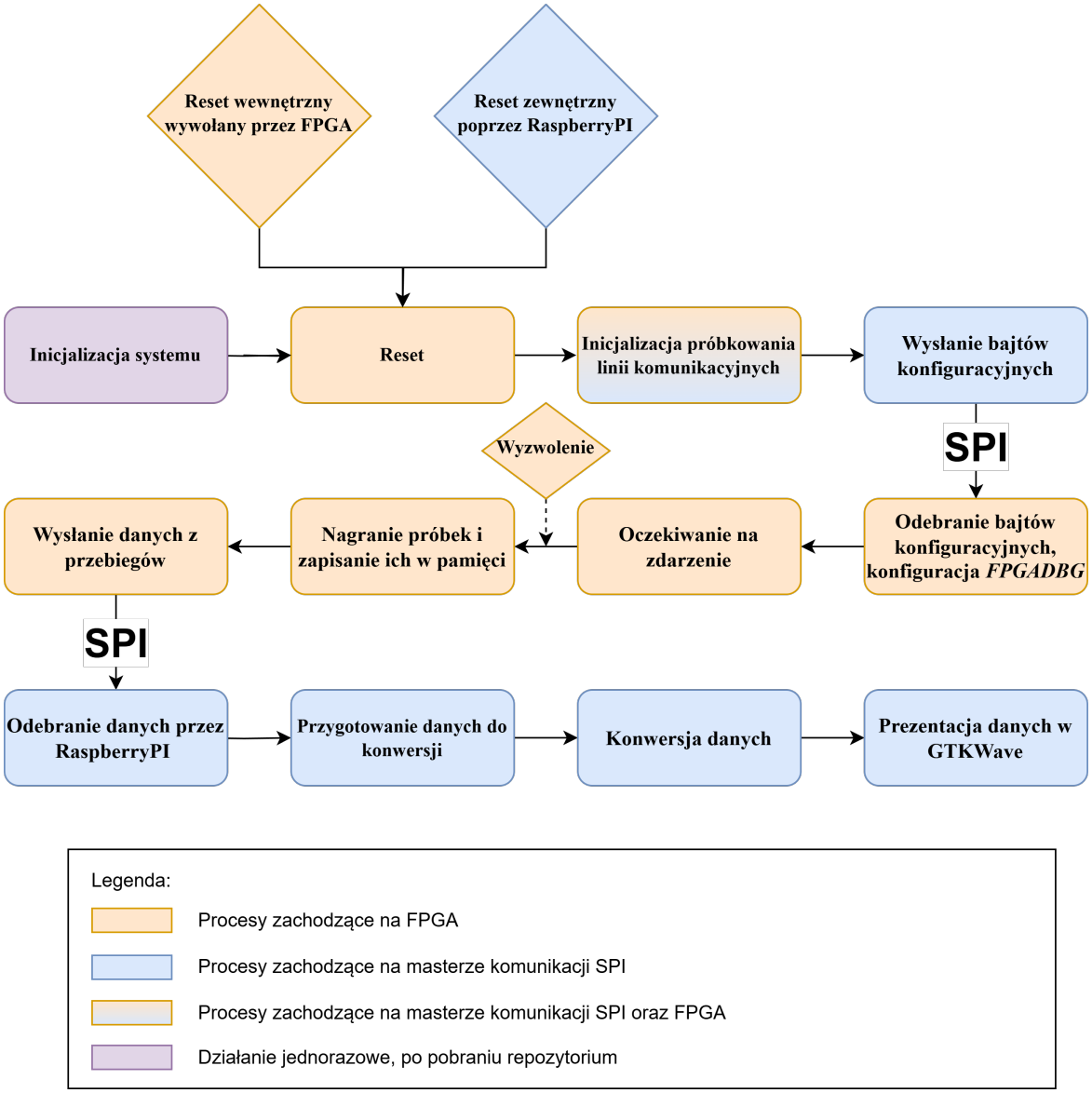
```
SPI_bus = 0
SPI_chip_select = 0

#default SPI_bus 0, chip select 0 pins
#SPI_MOSI_PIN = 10  # GPIO 10, Pin 19
#SPI_MISO_PIN = 9   # GPIO 9, Pin 21
#SPI_SCLK_PIN = 11  # GPIO 11, Pin 23
#SPI_CS_PIN_0 = 8     # GPIO 8, Pin 24 (CS0)
#SPI_CS_PIN_1 = 7     # GPIO 7, Pin 26 (CS1)

spi = spidev.SpiDev()
spi.open(SPI_bus, SPI_chip_select)  # Open bus 0, chip select 0

spi.mode = 0b00  # Set SPI mode (0 to 3)
```

```
spi.max_speed_hz = 5400000  # Set SPI speed
```

W przypadku chęci korzystania z innych linii komunikacyjnych należy odpowiednio ustawić wartości `SPI_bus` oraz `SPI_chip_select`. Przy wykorzystaniu magistrali SPI 1 należy wykorzystać inne piny do podłączenia z FPGA:

- MOSI - GPIO 20, Pin 38
- MISO - GPIO 19, Pin 37
- SPI CLOCK - GPIO 21, Pin 40
- CS0 - GPIO 16, Pin 36

5.2. Proces działania



Rysunek 5.3. Schemat blokowy działania układu.

1. **Inicjalizacja systemu** - zainstalowanie wymaganych bibliotek oraz oprogramowania. Następnie odpowiednio skompilowanie ich lub do ścieżek systemowych. *Ten krok należy wykonać tylko raz - przed pierwszym uruchomieniem narzędzia. Potem należy go pominąć.*
2. **Reset** - przywrócenie domyślnych stanów poszczególnych modułów FPGA oraz ich zmiennych do domyślnych wartości. Reset jest wywołany przez odczyt stanu GPIO. Należy wyprowadzić zdefiniować wejście w pliku constrain. W zależności od potrzeby projektu, zostały udostępnione dwie opcje podłączenia wejścia:
   - **Reset wewnętrzny wywołany przez SoC** - jeśli układ umożliwia reset za pomocą guzika, przełącznika, lub innego łatwo dostępnego peryferium.

- **Reset zewnętrzny poprzez RaspberryPI** - reset wystawiony za pomocą na RaspberryPI (*domyślnie ustawiony na GPIO 3 RaspberryPI*).

3. **Inicjalizacja próbkowania linii komunikacyjnych** - linie *MOSI* oraz *SPIclock* między modułem głównym i podrzędnym są próbkowane za pomocą 2-bitowego samplera, używającego decyzji większościowej. Pozwala to na dokładniejsze próbkowanie linii komunikacyjnych eliminując potencjalne przekłamania wynikające z pojedynczych zmian wartości bitów w sygnale wynikających z szumów. Sampler jest włączany za pomocą CE. W przypadku stanu wysokiego, sampler zwraca zawsze wartość 1 i odczyt danych konfiguracyjnych ani zegara SPI się nie odbywa. Umożliwia to podłączenie kilku układów FPGA równolegle do siebie. Korzystanie z samplera wymusza maksymalną wartość taktowania *SPIclock*:

$$\text{SPIclock}_{\text{frequency}} < \frac{\text{FPGAclock}_{\text{frequency}}}{3}$$

4. **Wysłanie bajtów konfiguracyjnych** - RaspberryPI wysyła do układu FPGA 3 bajty konfiguracyjne, które definiują ilość zapisanych w pamięci próbek po wyzwoleniu zdefiniowanym przez użytkownika oraz wybraną konfigurację, która pozwala na zdefiniowanie kilku zdarzeń, bądź różnych nagrywanych sygnałów bez konieczności ponownej syntezy, implementacji oraz wgrania strumienia bitowego.
5. **Odebranie bajtów konfiguracyjnych, konfiguracja *FPGADBG*** - bajty konfiguracyjne są opisane w poniższy sposób:

Byte number	b[7]	b[6]	b[5]	b[4]	b[3]	b[2]	b[1]	b[0]
1st byte	**(reserved)**	**(reserved)**	t[5]	t[4]	t[3]	t[2]	t[1]	t[0]
2nd byte	**(reserved)**	**(reserved)**	t[11]	t[10]	t[9]	t[8]	t[7]	t[6]
3rd byte	**(reserved)**	x[2]	x[1]	x[0]	t[15]	t[14]	t[13]	t[12]

Tabela 5.1. Znaczenie bitów w bajtach konfiguracyjnych

Legenda:

- **b[n]** - bajt przesłany przez SPI od mastera.
- **t[n]** - 16 bitowa zmienna określająca ilość próbek, które mają zostać zapisane. Maksymalna wartość to 65536.
- **x[n]** - 3 bity umożliwiające debugowanie 8 różnych części projektu, poprzez uzależnienie zapisywanych sygnałów od wartości zmiennej a także uzależnienie rozpoczęcia rejestracji przebiegów od różnych zdarzeń wyzwalających bez potrzeby ponownej syntezy, implementacji oraz wgrywania ciągu bitowego na FPGA.

6. **Oczekiwanie na zdarzenie** - *wyzwolenie powinno być wywołane przez użytkownika po spełnionym warunku, np. gdy badana zmienna osiągnęła określoną wartość.*

7. **Nagranie próbek i zapisanie ich w pamięci** - po zdarzeniu, próbki są zapisywane w pamięci BRAM o typie TDPRAM.
8. **Wysłanie danych z przebiegów** - w pierwszej kolejności wysłane sa bajty z zakodowanymi informacjami o przebiegu:
   A. `log2samples` - bajt informujący o maksymalnej liczbie próbek, które mogły zostać nagrane.
   B. `width` - bajt informujący o łącznej długości bitowej nagrywanych sygnałów.
   C. `number_of_signals` - bajt informujący o deklarowanej liczbie badanych sygnałów.
   D. `trigger_position` - 1 lub 2 bajty informujące o momencie wyzwolenia. Transmitowane są metodą LSB first.
   E. `stop_position` - 1 lub 2 bajty informujące o momencie wpisania ostatnie bitu do bufora. Transmitowane są metodą LSB first.

   Następnie są wysyłane nagrane przebiegi w liczbie określonej przez wysłane wcześniej bajty konfiguracyjne metodą LSB first.
9. **Odebranie danych przez RaspberryPI** - Master odbiera dane i zapisuje w tabeli krotek (eng. tuple).
10. **Przygotowanie danych do konwersji** - Usunięcie z tabeli niepotrzebnych:
    - **początkowy szum,**
    - **bity, czytane z linii podczas wczytywania danych z pamięci.**

    Następuje weryfikacja poprawności otrzymanych danych oraz zapisanie ich jako ciąg bitów.
11. **Konwersja danych** - dane są dekodowane, wartości log2samples, width, trigger position oraz stop position są odszyfrowywane. Następnie generowany jest plik *.lxt*, zawierający odczytane dane. Służy on do prezentacji danych za pomocą programu GTKWave.
12. **Prezentacja danych w GTKWave** - dane są gotowe do debugowania, ich podgląd jest udostępniony dla użytkownika za pomocą aplikacji GTKWave.

6. Diagnostyka wybranych systemów cyfrowych

6.1. Układy FPGA użyte do implementacji testowej

Wykonano implementację systemu **FPGADBG** na trzech różnych systemach:

Tabela 6.1. Systemy SoC, na których zaimplementowano projekt.

Płytka rozwojowa	FPGA	Producent
PYNQ-Z2[48]	ZYNQ XC7Z020-1CLG400C[49]	AMD
Tang Nano 20K[50]	GW2AR-18 QN88[19]	Sispeed, Gowin
DE0-Nano-SoC Kit/Atlas-SoC Kit[51]	Cyclone V[52]	terasIC, Intel®

6.2. Testy funkcjonalne

6.2.1. Testy integracyjne

Wykonano oddzielne testy integracyjne części fpgadbg core oraz fpgadbg conv. Oba testy zostały przeprowadzone poprzez ich symulację za pomocą danych, które mogły odpowiadać faktycznemu działaniu programu.

```
entity test_bit_transfer is
end test_bit_transfer;
architecture behavior of test_bit_transfer is
    -- Component declaration for the Unit Under Test (UUT)
    component pynq
        port(
            CE    : in  std_logic;
            SPI_clock : in  std_logic;
            MOSI  : in  std_logic;
            sys_rst : in  std_logic;
            sys_clk : in  std_logic
        );
    end component;
        -- Signals to connect to the UUT
    signal CE       : std_logic := '0'; -- CE is always 0 during transmission
    signal SPI_clock : std_logic := '0';
    signal MOSI     : std_logic := '0';
    signal sys_rst  : std_logic := '0';
    signal sys_clk : std_logic := '0'
    -- Clock period constant
    constant CLOCK_PERIOD : time := 20 ns;
    constant CLOCK_PERIOD_SPI : time := 100 ns;
    signal c1: std_logic_vector(7 downto 0) := "00000100";
    signal c2: std_logic_vector(7 downto 0) := "01011100";
    signal c3: std_logic_vector(7 downto 0) := "10000000";

begin
    -- Instantiate the Unit Under Test (UUT)
    uut: fpgadbg_s3sb
        port map (
            CE        => CE,
            SPI_clock => SPI_clock,
            MOSI      => MOSI,
            sys_rst   => sys_rst,
            sys_clk => sys_clk
        );
    -- Clock process for generating SPI clock
    clock_process : process
    begin
        while true loop
            SPI_clock <= '0';
            wait for CLOCK_PERIOD_SPI / 2;
            SPI_clock <= '1';
            wait for CLOCK_PERIOD_SPI / 2;
        end loop;
    end process clock_process;
    second_clock_process  : process
    begin
        while true loop
            sys_clk <= '0';
            wait for CLOCK_PERIOD / 2;
            sys_clk <= '1';
            wait for CLOCK_PERIOD / 2;
        end loop;
    end process second_clock_process ;
    -- Test process to send c1, c2, and c3 over MOSI
    stimulus_process: process
    begin
        sys_rst <= '0';
        wait for 5 * CLOCK_PERIOD;
        sys_rst <= '1';
        wait for 5 * CLOCK_PERIOD;
        -- Send data over MOSI
        for i in 7 downto 0 loop
            MOSI <= c1(i);
            wait for CLOCK_PERIOD_SPI;
        end loop;
        for i in 7 downto 0 loop
            MOSI <= c2(i);
            wait for CLOCK_PERIOD_SPI;
        end loop;
        for i in 7 downto 0 loop
            MOSI <= c3(i);
            wait for CLOCK_PERIOD_SPI;
        end loop;
        -- Wait for a few clock cycles and end simulation
        wait for 1000000 * CLOCK_PERIOD;
        assert false report "Simulation ended" severity failure;
    end process stimulus_process;
end behavior;
```

Listing 1. Plik symulacyjny służący do testowania działania fpgadbg core.

Po poprawnym zdekodowaniu metadanych [Listing 3], program tworzy plik *.lxt*, który wyświetli stałą wartość badanego sygnału [Listing 2].

```
import os
import spidev
import sys
import time
import fpgadbg_conv

data = "1000101000100001011000010000000101100011000000011011111000000001000000110000000000000011011111000000001000000110000000000000001"
assign=[(32,0,"test_value")]
# Create the data converter, and create the LXT file
cnv=fpgadbg_conv.fpgadbg_conv(assign,8,125,-10,"test.lxt")
cnv.conv(data)
# Display the waveforms if gtkwave is available
os.system("gtkwave test.lxt test.sav")
```

Listing 2. Plik symulacyjny do testowania fpgadbg conv.

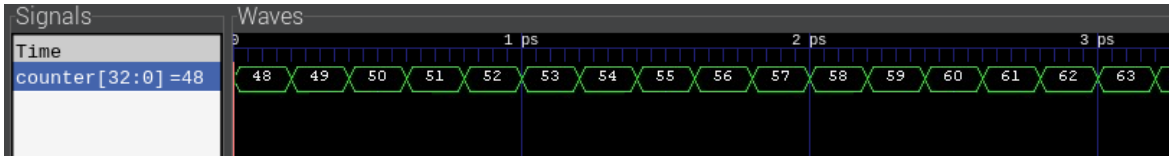
```
word_len = self.nbits

# Extract log2samples (first 8 bits) and convert to integer

log2samples = int(bit_string[4:8], 2)
if(str(bit_string[0:3]) == "100"):
    filled = 1
else:
    filled = 0
num_of_samples = 1 << log2samples

# Extract data_width (next 8 bits) and convert to integer
self.data_width = int(bit_string[8:16], 2)
# Calculate the number of data words per sample
self.words_per_sample = (self.data_width + word_len - 1) // word_len
# Extract trigger and stop positions based on data_width
if word_len >= 16:
    trig_pos = int(bit_string[16:32], 2)
    stop_pos = int(bit_string[32:48], 2)
    first_data = 48  # Start index of actual data in the bit string
else:
    trig_pos = int(bit_string[16:24], 2) + 256 * int(bit_string[24:32], 2)
    stop_pos = int(bit_string[32:40], 2) + 256 * int(bit_string[40:48], 2)
    first_data = 48
```

Listing 3. Fragment kodu służący do zdekodowania metadanych.

6.2.2. Testy systemowe

Działanie całego narzędzia jako całości przetestowano na algorytmach danych wejściowych jako badanych sygnałach:

- stała liczba,
- licznik,
- miganie.

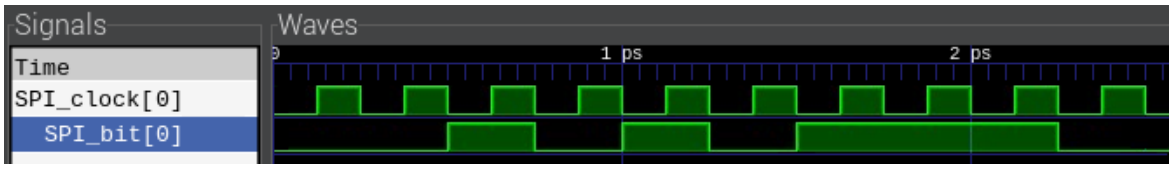


Rysunek 6.1. Zarejestrowany przebieg licznika.

6.2.3. Testy użytkowe

Test użytkowy wykonano integrując **Przenośny system do diagnostyki i uruchamiania systemów cyfrowych realizowanych w układach FPGA** do projektu obsługi wyświetlacza OLED w układzie PmodOLED[53][54] firmy Digilent. Wykorzystano do tego płytkę rozwojową PYNQ-Z2 [48].

6.3. Badania wykorzystania zasobów systemu



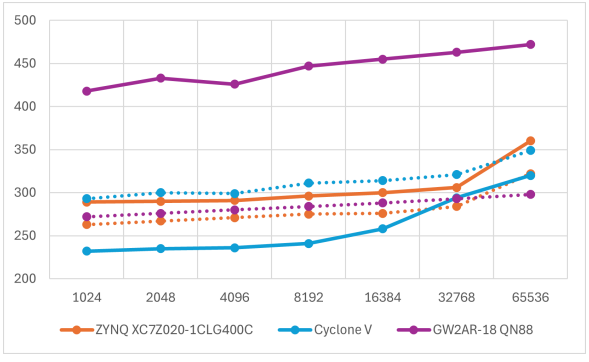
Rysunek 6.2. Zarejestrowane przebiegi wewnętrznego zegaru SPI clock oraz bitu SPI wysyłanego do bufora w ramach inicjalizacji ekranu. Widoczna komenda to 0xAE.

Tabela 6.2. Porównanie zużycia zasobów FPGA w projekcie obsługi wyświetlacza OLED w układzie PmodOLED [53][54] firmy Digilent dołączonego do płytki rozwojowej PYNQ-Z2 [48] przy szerokości sygnału wejściowego 10 bitów oraz nagranych 65536 próbkach.

Zasób	**Brak fpgadbg**	**Zaimplementowane fpgadbg**
LUT	891	1183
FF	1244	1505
BRAM	0	20
URAM	0	0
DSP	0	0
WNS[ns]	11,811	11,479
WHS[ns]	0,013	0,036
WPWS[ns]	9,500	9,500
Całkowity pobór mocy [mW]	99	108

Tabela 6.3. Porównanie zużycia zasobów dla wybranej ilości próbek w przypadku badania sygnałów wejściowych o szerokości łącznej 33 bitów.

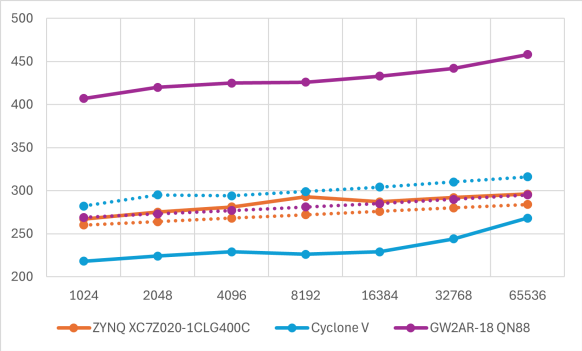
Zasoby** / FPGA	**1024**			**16384**			**65536**		
	LUT	FF	Moc	LUT	FF	Moc	LUT	FF	Moc
ZYNQ XC7Z020-1CLG400C[49]	293	263	109	300	276	114	359	336	125
GW2AR-18 QN88[19]	418	272	144	455	288	260	472	298	654
Cyclone V[52]	232	293	*	258	314	*	320	349	*



Rysunek 6.3. Wykres w zużycia zasobów przez Przenośny system do diagnostyki i uruchamiania systemów cyfrowych realizowanych w układach FPGA w przypadku badania sygnałów wejściowych o szerokości łącznej 33 bitów. Liniami przerywanymi oznaczono zużycie FF, a ciągłymi wykorzystanie LUT.

Tabela 6.4. Porównanie zużycia zasobów dla wybranej ilości próbek w przypadku badania sygnałów wejściowych o szerokości łącznej 8 bitów.

Zasoby** / FPGA	**1024**			**16384**			**65536**		
	LUT	FF	Moc	LUT	FF	Moc	LUT	FF	Moc
ZYNQ XC7Z020-1CLG400C[49]	267	260	108	287	276	108	296	284	109
GW2AR-18 QN88[19]	407	269	142	433	285	181	458	295	339
Cyclone V[52]	218	282	*	229	304	*	268	316	*



Rysunek 6.4. Wykres w zużycia zasobów przez Przenośny system do diagnostyki i uruchamiania systemów cyfrowych realizowanych w układach FPGA w przypadku badania sygnałów wejściowych o szerokości łącznej 8 bitów. Liniami przerywanymi oznaczono zużycie FF, a ciągłymi wykorzystanie LUT.

* Quartus Lite nie udostępnia możliwości analizy zużycia mocy [55]. Ze względu na cenę wersji Pro, zdecydowałem się nie wykonywać tych testów.

** Moc jest podawana w miliwatach z dokładnością do 1 miliwata.

Testy wykazały, że nie każda FPGA radzi sobie równie dobrze z optymalizacją wykorzystania zasobów w przypadku korzystania z przenośnego systemu do diagnostyki i uruchamiania systemów cyfrowych realizowanych w układach FPGA. Środowisko gowin EDA [19] optymalizując projekt na FPGA GW2AR-18 QN88[19] poradziło sobie najgorzej. Był to wynik przewidywalny, ze względu na dużo wyższe ceny, a co za tym idzie jakość produktu konkurentów. ZYNQ XC7Z020-1CLG400C[49] oraz Cyclone V[52] osiągneły podobne wyniki. Warto zwrócić uwagę na to, że wykorzystanie LUT oraz FF FPGA, przy znacznym wzroście ilości badanych próbek, nie wzrasta drastycznie. Warto rozważyć korzystanie z jak największej ilości zapisanych próbek, aby zobaczyć jak największą część badanego sygnału.

7. Podsumowanie i wnioski

7.1. Podsumowanie pracy

Przenośny system do diagnostyki i uruchamiania systemów cyfrowych realizowanych w układach FPGA jest uniwersalnym narzędziem, dzięki któremu, po krótkim zapoznaniu, łatwo debugować różne systemy FPGA, niezależnie od architektury producenta. Można go stosować jako otwarty i udostępniony na zasadzie wolnej licencji IP core pozwalający na debugowanie FPGA.

7.2. Wnioski końcowe

Testy wykazały, że nie każda FPGA radzi sobie równie dobrze z optymalizacją wykorzystania zasobów w przypadku korzystania z przenośnego systemu do diagnostyki i uruchamiania systemów cyfrowych realizowanych w układach FPGA. W przypadku korzystania z projektu oraz dalszego rozwoju należałoby wziąć pod uwagę ograniczenia producentów. W przypadku dołączenia go do projektu, należy zbadać, czy na pewno zmieści się on w zasobach FPGA. Nie zajmuje on dużo miejsca, jednak gdy zbliżamy się do maksymalnego wykorzystania zasobów przez połączenie z innymi modułami, optymalizacja może być gorsza. Dodatkowo należy rozważyć możliwe komplikacje, rzadziej spotykane w przypadku debugowania za pomocą tradycyjnych metod. Takimi komplikacjami mogą być między innymi:

- nie działający pin na RaspberyPI,
- przerwany przewód między masterem i slavem,
- brak internetu (w przypadku korzystania z SSH).

Nie zmienia to faktu, że jest to narzędzie łatwo przenośne i uniwersalne. Warto rozważyć jego zastosowanie, jeśli działamy na wielu różnych FPGA, korzystających z różnych środowisk programistycznych.

7.3. Propozycje dalszego rozwoju

Rozwój przenośnego systemu do diagnostyki i uruchamiania systemów cyfrowych realizowanych w układach FPGA jest możliwy w wielu różnych kierunkach:

- **Obsługa dodatkowych FPGA** - Dodanie dodatkowych FPGA, wykorzystując istniejące szablony zwiększy uniwersalność rozwiązania.
- **Dodanie dodatkowego interface'u komunikacyjnego** - wrapper wykorzystujący JTAG, lub SWD mógłby zapewnić większe możliwości. Warto rozważyć wykorzystanie OpenOCD.
- **Wykorzystanie chmury** - pobrane dane z FPGA mogłyby być od razu udostępniane w chmurze z możliwością późniejszego pobrania.

- **Dodanie możliwości syntezy na poziomie RaspberryPI** - Za pomocą innych open-source'owych rozwiązań można byłoby umożliwić użytkownikowi syntezę, implementację i wgranie bitstremu na wybrane architektury FPGA.
- **Przekazywanie informacji o debugowanych sygnalach bezpośrednio z FPGA** - obecnie użytkownik musi zadeklarować w module głównym VHDL zmienne, które chce debugować, a potem zrobić to jeszcze raz w plikach python. Należałoby ten proces uprościć, żeby wykluczyć możliwe pomyłki.
- **Zbadanie poboru mocy na Cyclone V FPGA za pomocą Quartus Pro**

Kod źródłowy został udostępniony pod licencją GNU General Public License, *Free Software Foundation.*

Bibliografia

[1] *Vivado Design Suite User Guide Programming and Debugging UG908 (v2022.1) April 26, 2022e*, AMD, 2022.

[2] *Intel® Quartus® Prime Standard Edition User Guide: Debug Tools*, Intel®, 2018.

[3] dr hab. inż. Wojciech Zabołotny, "Fpgadbg - a tool for FPGA debugging", Dostęp zdalny (25.11.2024): https://koral.ise.pw.edu.pl/~wzab/fpgadbg/, 2006.

[4] Markets i Markets, "Field Programmable Gate Array (FPGA) Market Size, Share Industry Trends Analysis Report by Configuration (Low-end FPGA, Mid-range FPGA, High-end FPGA), Technology (SRAM, Flash, Antifuse), Node Size (=16 nm, 20-90 nm, >90 nm), Vertical (Telecommunications, Data Center Computing, Automotive) Region - Global Forecast to 2029", Dostęp zdalny (25.11.2024): https://www.marketsandmarkets.com/Market-Reports/fpga-market-194123367.html, 2024.

[5] hardwarebee, "List of FPGA Companies", Dostęp zdalny (26.11.2024): https://hardwarebee.com/list-fpga-companies/, 2018.

[6] J. Schneider i I. Smalley, "What is a field programmable gate array (FPGA)?", Dostęp zdalny (25.11.2024): https://www.ibm.com/think/topics/field-programmable-gate-arrays, 2024.

[7] P. zbiorowa pod redakcją Grzegorza Karpiela, *Układy FPGA w systemach sterowania i regulacji*. Akademia Górniczo-Hutnicza w Krakowie: Katedra Robotyki i Mechatroniki, 2023.

[8] N. Instruments®, "FPGA Fundamentals: Basics of Field-Programmable Gate Arrays", Dostęp zdalny (25.11.2024): https://www.ni.com/en/shop/electronic-test-instrumentation/add-ons-for-electronic-test-and-instrumentation/what-is-labview-fpga-module/fpga-fundamentals.html, 2022.

[9] D. Benson, "Getting Started with FPGAs: Lookup Tables and Flip-Flops", Dostęp zdalny (25.11.2024): https://www.allaboutcircuits.com/technical-articles/getting-started-with-fpgas-look-up-tables-and-flip-flops/, 2017.

[10] P. Kallstrom, "Simplified example illustration of a logic cell (LUT – Lookup table, FA – Full adder, DFF – D-type flip-flop)", Dostęp zdalny (25.11.2024): https://en.wikipedia.org/wiki/Field-programmable_gate_array#/media/File:FPGA_cell_example.png, 2010.

[11] A. SADAY, "A REVIEW OF FPGA-BASED APPLICATIONS AND FPGA USAGE IN THE INDUSTRIAL AREA", w: *Innovations and technologies in engineering*, Eğitim Yayınevi, 2022, s. 171–183.

[12] K. Sungho, "High-Speed Incoming Infrared Target Detection by Fusion of Spatial and Temporal Detectors", *Frontiers in Infrared Photodetection*, t. EISSN 1424-8220, nr 15(4), s. 7267–7293, 2015.

[13] AMD, *Aerospace Defense Solutions, Space*, Dostęp zdalny (26.11.2024): https://www.amd.com/en/solutions/aerospace-and-defense/space.html, 2024.

[14] T. Wu, W. Liu i Y. Jino, "An End-to-End Solution to Autonomous Driving Based on Xilinx FPGA", w: *2019 International Conference on Field-Programmable Technology (ICFPT)*, IEEE, 2019.

[15] 3BL, "Road Experience Management targets accuracy to enable fully autonomous driving", Dostęp zdalny (25.11.2024): https://www.3blmedia.com/news/gm-exploring-mobileye-advanced-mapping-onstar-data, 2016.

[16] MarketsUS, "Global Field-Programmable Gate Array (FPGA) Market By Configuration (Low-Range FPGA, Mud-End FPGA, and High-End FPGA), By Technology (SRAM, Flash, Antifuse, and Others), By Node Size, By End Use Industry, By Region and Companies - Industry Segment Outlook, Market Assessment, Competition Scenario, Trends, and Forecast 2023-2032", Dostęp zdalny (25.11.2024): https://market.us/report/fpga-market/, 2024.

[17] *Quartus® Prime Pro Edition User Guide: Getting Started*, Intel®, 2024.

[18] *Lattice Diamond Software documentation*, Lattice Semiconductor, 2019.

[19] G. Semiconfuctors, *Rodzina Chenxi*, Dostęp zdalny (19.01.2024): https://www.gowinsemi.com.cn/prod_view.aspx?TypeId=10&FId=t3:10:3&Id=167, 2024.

[20] *Słownik języka polskiego - definicja słowa 'diagnostyka'*, Dostęp zdalny (14.03.2024): https://sjp.pwn.pl/slowniki/diagnostyka, 2024.

[21] *Xylinx*, Dostęp zdalny (11.01.2025): https://docs.amd.com/v/u/en-US/ds875-ila, AMD, 2012.

[22] *Reveal User Guide*, Dostęp zdalny (11.01.2025): https://www.latticesemi.com/~/media/328D471BF2C74EB1907832FAA6FB344B.ashx, Lattice Semiconductors, 2015.

[23] *Gowin Analyzer Oscilloscope User Guide*, Dostęp zdalny (11.01.2025): https://www.gowinsemi.com/upload/database_doc/37/document/5bfcfed078251.pdf, Gowin Semiconductor, 2018.

[24] D. Rath, *OpenOC*, Dostęp zdalny (11.01.2025): https://openocd.org/doc-release/html/index.html, 2024.

[25] U. Hermann, *The sigrok project*, Dostęp zdalny (12.01.2025): https://sigrok.org/wiki/Main_Page, 2023.

[26] U. Finkelstein, *GTKWave 3.3 Wave Analyzer User's Guide*, Dostęp zdalny (12.01.2025): https://gtkwave.sourceforge.net/gtkwave.pd, 2020.

[27] F. U. Analyzer, *HDD Software*, Dostęp zdalny (11.01.2025): https://freeusbanalyzer.com/, 2024.

[28] R. Sharpe, *Wireshark User's Guide*, Dostęp zdalny (11.01.2025): https://www.wireshark.org/docs/wsug_html_chunked/, 2024.

[29] E. Digital, *LiteX*, Dostęp zdalny (11.01.2025): https://github.com/litex-hub, 2025.

[30] E. Digital, *LiteScope*, Dostęp zdalny (11.01.2025): https://github.com/enjoy-digital/litescope, 2025.

[31] E. Digital, *LiteETH*, Dostęp zdalny (11.01.2025): https://github.com/enjoy-digital/liteeth, 2025.

[32] *OpenTitan Project*, lowRISC C.I.C., 2024.

[33] W. Labs, *Openbench Logic Sniffer*, Dostęp zdalny (12.01.2025): http://dangerousprototypes.com/docs/Open_Bench_Logic_Sniffer, 2017.

[34] *Questa® SIM User's Manual*, Dostęp zdalny (24.01.2025): https://ww1.microchip.com/downloads/aemDocuments/documents/FPGA/swdocs/questasim/questa_sim_user_2024_2.pdf, SIEMENS EDA, 2024.

[35] *Incisive Enterprise Simulator*, Dostęp zdalny (24.01.2025): http://pdf2.solecsy.com/565/c02013ae-ffce-4436-9866-182120ab263c.pdf, Cadence®, 2011.

[36] *VCS®/VCSi™ User Guide*, Dostęp zdalny (24.01.2025): https://users.ece.utexas.edu/~patt/10s.382N/handouts/vcs.pdf, Synopsys, 2008.

[37] cocotb contributors, *cocotb's documentation*, Dostęp zdalny (12.01.2025): https://docs.cocotb.org/en/stable/, 2024.

[38] W. Snyder, *Verilator User's Guide*, Dostęp zdalny (12.01.2025): https://verilator.org/guide/latest/, 2024.

[39] F4PGA, *Wymbiflow - open source flow for generating bitstreams from Verilog*, Dostęp zdalny (12.01.2025): https://github.com/symbiflow, 2023.

[40] K. E. Murray, T. R. Ansell, K. Rothman i A. Comodi, "Symbiflow VPR: An Open-Source Design Flow for Commercial and Novel FPGAs", Dostęp zdalny (26.11.2024): https://www.researchgate.net/publication/341717917_Symbiflow_VPR_An_Open-Source_Design_Flow_for_Commercial_and_Novel_FPGAs, 2020.

[41] Y. HQ, *Icestorm*, Dostęp zdalny (11.01.2025): https://github.com/YosysHQ/icestorm, 2024.

[42] RaspberryPI, *Raspberry PI documentation*, Dostęp zdalny (12.01.2025): https://www.raspberrypi.com/documentation/, 2025.

[43] T. Bybell, lxt_write, Dostęp zdalny (14.01.2024): https://github.com/Parrot-Developers/lttng2lxt/tree/master, 2005.

[44] Nandland, "What is a Block RAM (BRAM) in an FPGA?", Dostęp zdalny (25.11.2024): https://nandland.com/lesson-15-what-is-a-block-ram-bram/, 2022.

[45] D. Systems, "Memory Design", Dostęp zdalny (19.01.2024): https://digitalsystemdesign.in/wp-content/uploads/2018/05/Memory.pdf, 2018.

[46] V. Thoms, *spidev 3.6*, Dostęp zdalny (19.01.2024): https://pypi.org/project/spidev/, 2022.

[47] M. Grusin, "Serial Peripheral Interface (SPI)", Dostęp zdalny (12.01.2025): https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all, 2018.

[48] *PYNQ-Z2 Reference Manual v1.0*, Dostęp zdalny (19.01.2025): https://www.lcsc.com/datasheet/lcsc_datasheet_1912111437_AMD-XILINX-PYNQ-Z2_C393968.pdf, AMD, 2018.

[49] *Zynq-7000 SoC Data Sheet: Overview*, Dostęp zdalny (19.01.2025): https://docs.amd.com/v/u/en-US/ds190-Zynq-7000-Overview, AMD, 2018.

[50] Sispeed, *Tang Nano 20k*, Dostęp zdalny (19.01.2024): https://wiki.sipeed.com/hardware/en/tang/tang-nano-20k/nano-20k.html, 2023.

[51] terasIC, *DE0-Nano-SoC Kit/Atlas-SoC Kit*, Dostęp zdalny (19.01.2024): https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=167&No=941, 2024.

[52] *Cyclone V Device Datasheet*, Dostęp zdalny (19.01.2024): https://www.intel.com/programmable/technical-pdfs/683801.pdf, Intel®, 2023.

[53] *PmodOLED™ Reference Manual*, Dostęp zdalny (11.01.2025): https://digilent.com/reference/_media/reference/pmod/pmodoled/pmodoled_rm.pdf, Diligent®, 2016.

[54] *Digilent Pmod™ Interface Specification 1.2.0*, Dostęp zdalny (11.01.2025): https://digilent.com/reference/_media/reference/pmod/pmod-interface-specification-1_2_0.pdf, Diligent®, 2017.

[55] Intel®, *Quartus® Prime Design Software*, Dostęp zdalny (14.03.2024): https://www.intel.co.kr/content/dam/www/central-libraries/us/en/documents/quartus-prime-compare-editions-guide.pdf?, 2024.

Wykaz symboli i skrótów

ACAP – Adaptive Compute Acceleration Platforms
ASIC – Application-Specific Integrated Circuit
ASIC CAD – Application-Specific Integrated Circuit Computer-Aided Design
BRAM – Block Random-Access Memory
CE – Chip Enable
CLB – Configurable Logic Blocks
CS – Chip Select
DFF – D-type flip-flop
EBR – Embedded Block RAM
FA – Full Adder
FF – Flip-flop
FPGA – Field Programmable Gate Array
GPI – General Purpose Input
I2C – Inter-Integrated Circuit
IP – Intellectual Property
JTAG – Joint Test Action Group
LSB – Least Significant Bit
LUT – Look-Up Table
MISO – Master in Slave out
MOSI – Master out Slave in
MUX – Multiplexer
OCD – On-Chip Debugger
PC – Personal Computer
PCB – Printed Circuit Board
PICO – Peripherial in Controller out
POCI – Peripherial out Controller in
SDR – Software-Defined Radio
SJP – Słownik Języka Polskiego
SoC – System on Chip
SPI – Serial Peripheral Interface
SSH – Secure Shell
SWD – Serial Wire Debug
TCL – Tool Command Language
TDPRAM – True Dual-Port RAM
UART – Universal Asynchronous Receiver-Transmitter
USB – Universal Serial Bus
USG – Ultrasonografia
VHDL – Very High Speed Integrated Circuit Hardware Description Language

VPR – Versatile Place and Route
WHS – Worst Hold Slack
WNS – Worst Negative Slack
WPWS – Worst Pulse Width Slack

Spis rysunków

Spis tabel

Spis listingów

Spis załączników

Załącznik 1. Repozytorium github

https://github.com/PawelMurdzek/fpgadb3