Warsaw University of Technology

FACULTY OF
ELECTRONICS AND INFORMATION TECHNOLOGY

Institute of Institute of Electronic Systems

Bachelor's diploma thesis

in the field of study Electronics
and specialisation Electronics and Photonics

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

Paweł Murdzek
student record book number 310850

thesis supervisor
dr hab. inż. Wojciech Zabołotny

WARSAW 2025

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 presented 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 List of Listings

załącznik nr 10 do zarządzenia
nr 46 /2016 Rektora PW

Politechnika Warszawska
Warsaw University of Technology

……….....................
miejscowość i data
place and date

……………………………..
imię i nazwisko studenta
name and surname of the student

……………………………..
numer albumu
student record book number

……………………….………
kierunek studiów
field of study

OŚWIADCZENIE

DECLARATION

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.
Under the penalty of perjury, I hereby certify that I wrote my diploma thesis on my own, under the guidance of the thesis supervisor.

Jednocześnie oświadczam, że:
I also declare that:

- 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,
- *this diploma thesis does not constitute infringement of copyright following the act of 4 February 1994 on copyright and related rights (Journal of Acts of 2006 no. 90, item 631 with further amendments) or personal rights protected under the civil law,*
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- *the diploma thesis does not contain data or information acquired in an illegal way,*
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- *the diploma thesis has never been the basis of any other official proceedings leading to the award of diplomas or professional degrees,*
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- *all information included in the diploma thesis, derived from printed and electronic sources, has been documented with relevant references in the literature section,*
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.
- *I am aware of the regulations at Warsaw University of Technology on management of copyright and related rights, industrial property rights and commercialisation.*

Politechnika Warszawska
Warsaw University of Technology

załącznik nr 10 do zarządzenia
nr   /2016 Rektora PW

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.
I certify that the content of the printed version of the diploma thesis, the content of the electronic version of the diploma thesis (on a CD) and the content of the diploma thesis in the Archive of Diploma Theses (APD module) of the USOS system are identical.

....
czytelny podpis studenta
legible signature of the student

Contents

1. Introduction

1.1. Thesis Goal

The aim of this thesis is to create a universal system that will allow the user for repeatable and easy verification of projects created in hardware description language. This verification will take place in real time on the tested hardware, not in a simulation environment.

1.2. Project Scope

The designed open-source tool will utilize other open-source tools. Using it, we should be able to validate and run FPGAs from various distributors, without being forced to rely on environments prepared by them. Due to the assumption of system portability, the solution should be as universal and simple as possible. Due to the open nature of this project, it is also worth paying attention to the price of the solution so that it does not become an obstacle in using the system in practice.

1.3. Methodology

1.3.1 Market Analysis:

a) Research of solutions made available by current and past developers [1][2][3].

b) Checking the share of individual suppliers to determine the priority of development of subsequent solutions [4]

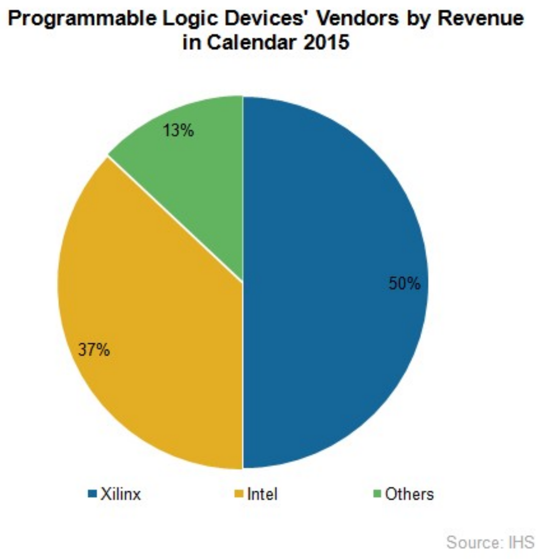

Figure 1.1. FPGA market share of individual manufacturers [5].

1.3.2 Execution of the debugger project with preliminary assumptions chosen after the market analysis described in point 1.a). Limiting to one SoC board per manufacturer.

1.3.3 Verification of project operation on FPGAs with different architectures.

1.3.4 Adding subsequent FPGA chips from the same manufacturers or adding other manufacturers.

1.3.5 General expansion of the project with additional methods enabling more convenient analysis of signal waveforms in the tested modules.

2. FPGA Chips and Their Application

2.1. Introduction to FPGA

FPGA - field programmable gate array are universal integrated circuits composed of logic gates that allow for easy creation of solution prototypes without the need to change the hardware itself. Unlike traditional ASIC - application-specific integrated circuit, they allow for multiple changes within the chip by modifying the system itself, which is done by programming a given board in a hardware description language [6]. FPGA chips ensure greater versatility due to GPIO ports, which in majority can be freely configurable effectively, unlike dedicated ports with little possibility of changing functionality. Additionally, they allow for performing multiple tasks simultaneously and in parallel, unlike switching between processes as a result of interrupts [7].

2.2. FPGA Architecture

FPGA chips consist of dozens, and in complex systems, up to several thousand matrix-distributed logic blocks *(CLB - configurable logic blocks)* as well as connections between these blocks and input/output ports, allowing for system communication with external circuits and peripherals.

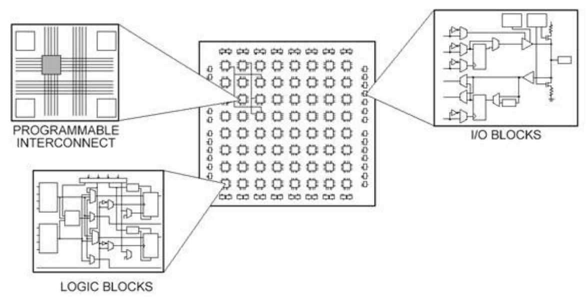

Figure 2.1. General scheme of an FPGA chip [8].

Individual matrix-distributed logic blocks consist of LUTs *(look-up tables)*, which serve as diverse logic blocks, and flip-flops which are used as counters, dividers, and memory cells [9].

2.3. FPGA Applications

To understand the application of FPGA, one must pay attention to their characteristics. Main features of the chips include, among others:

- Low latency allowing for convenient application for signal processing in real time.
- Rapid introduction of new solutions and hardware modifications by changing the hardware description language.
- Wide adaptive properties of FPGA to individual needs unlike traditional ASIC [7].

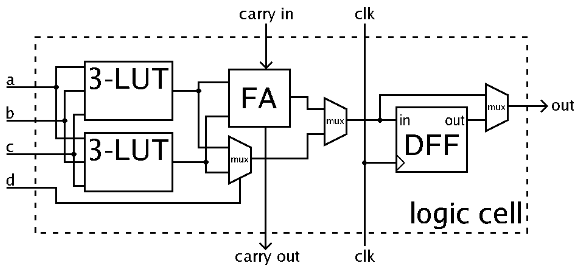

Figure 2.2. Example illustration of a CLB (LUT – Lookup table, FA – Full adder, DFF – D-type flip-flop) [10].

- Possibility of performing many operations in parallel during data processing, without using interrupts [7].
- Long "life" of the chip [11].
- In specific cases, they can ensure higher data processing speed than traditional GPUs while consuming less energy [11].

Using FPGA, however, involves a long design process, and also requires special expert skills and knowledge of the device architecture.

2.3.1. Classification by Industry

Main centers of FPGA use in industry are [11]:

1. **Space, Aerospace, and Military Industry** - due to high radiation resistance, they find many applications in signal processing, image processing [12], and in SDR [13]. Satellite communication systems and secure communication systems require simultaneous processing of large amounts of data, which is an additional argument for using reconfigurable chips.
2. **Automotive and Mobility** - due to low latency times in FPGA chips, they find application in automatic control systems. Additionally, the popularity of using FPGA in car battery management systems is growing [14].
3. **Robotics and Medicine** - parallel processing of multiple data allows for more accurate management of processes, such as for example image processing in USG examinations or in calculating optimal rotation of an axis working on production of a mechanical arm.
4. **Communication** - due to small latencies and the possibility of implementation enabling high data throughput, FPGAs are regularly used in the telecommunications industry. Companies such as Google, Microsoft, and Amazon use reconfigurable chips in their servers to accelerate data processing.

2.3.2. Most Popular Ways of Storing Configuration Data

The Markets.us report [16] lists three used technologies:

- **SRAM** - the most frequently used method of FPGA configuration, due to the most

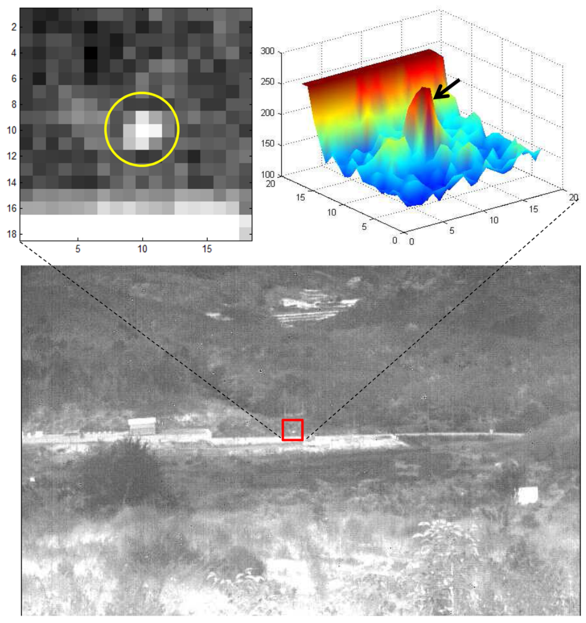

Figure 2.3. Use of FPGA in detecting small objects using infrared sensors [12].

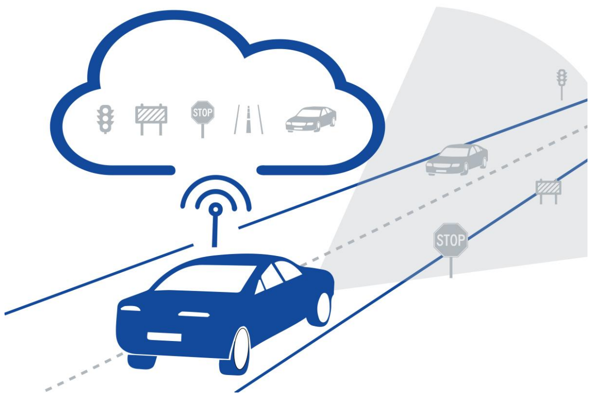

Figure 2.4. A driving car collects a large amount of data that must be processed quickly and accurately [15].

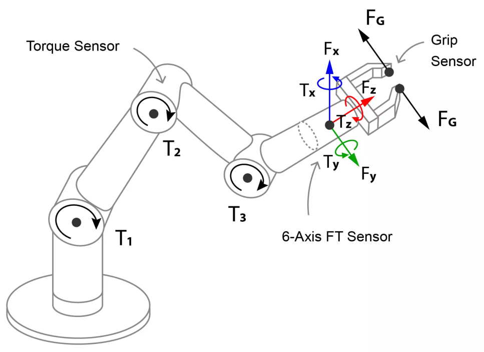

Figure 2.5. Simplified diagram of forces acting on a robot arm during work requiring synchronous multi-vector calculations.

universal and reprogrammable character. Finds applications among others in the automotive industry and data centers.

- **Flash** - due to lower energy consumption, they find the most applications in consumer electronics using batteries as the main power source. These include, among others, mobile phones.
- **Antifuse** - due to only a single programming possibility, longer life, and security resulting from the lack of reconfiguration capability, they find the most applications in the space, aerospace, and military industries.

2.4. Typical Tools for Creating Software for FPGA

Due to the difference in FPGA architecture, individual manufacturers provide their own software serving for optimal planning of the layout of used CLBs and connections between them in their reconfigurable chips. This is important for the most optimal use of FPGA resources *(LUT and FF)*, while not introducing errors in planned tasks and in processed data. In the FPGA environment, there is a race in which manufacturers try to accustom the consumer to their software, as well as FPGA, ensuring chips adapted to various applications. Consequently, everyone tries to lock designers using reconfigurable chips in their ecosystem and discourage using competitor products. Example programs serving for synthesis, implementation, and bitstream generation for FPGA:

- **Vivado Design Suite** - System from AMD optimized for complex chips from series 7, Ultrascale, Ultrascale+, Zynq-7000, as well as the novelty - Versal ACAP [1], adapted for AI development in cooperation with **Vitis** software.
- **Quartus® Prime** - Tool from Intel® serving for work with FPGA from Agilex, Stratix, Arria, and Cyclone series. It cooperates with many other tools of this company, serving for niche applications such as power consumption calculation using **Intel® FPGA Power and Thermal Calculator** [17].
- **Lattice Diamond** - Supports most FPGA series developed by Lattice Semiconductors, including MachXO, MachXO2, ECP2, ECP3, ECP5, CrossLink, and ispMACH 4000. Working with FPGA from iCE40, iCE40 UltraPlus, and CrossLink-NX series, we will use different software, optimized for design with consideration of energy consumption minimization [18].
- **Gowin EDA** - Tool from Gowin Semiconductor. Enables design of modules and their implementation on FPGA of Arora V, GW1NZ, and LittleBee families [19]. Some of them are characterized by significantly lower costs compared to competition. Their lower price is related to a smaller amount of resources available on the FPGA.

3. Digital Systems Diagnostics

3.1. The Concept of Diagnostics in Digital Systems

The Dictionary of Polish Language defines the concept of diagnostics *as determining the technical state of devices and determining the sources of failure*[20]. In the case of FPGA, this means verifying and validating the correctness of the process described on the reconfigurable board. For FPGA configuration, we use one of the hardware description languages such as Verilog or VHDL. Verification of the correctness of module operation can be carried out by simulating signal waveforms or observing these waveforms in real time directly on the tested FPGA. Specialized tools are used for this, allowing for the specification of trigger conditions of interest, after which waveforms are registered to memory. Detecting errors during module design, as well as after synthesis and implementation, is an inseparable part of programming. We colloquially call it **debugging**.

3.2. Popular Digital Systems Diagnostics Systems

3.2.1. Tools Prepared by Manufacturers

Every manufacturer, in addition to programming tools, also provides tools for diagnostics or debugging of projects. AMD, Intel®, and Lattice provide them as an extension to the FPGA design tool, respectively:

1. **Chipscope** - a tool proprietary to AMD for debugging signals directly on the reconfigurable chip [21]. In newer FPGA families, it has been replaced by **Vivado Integrated Logic Analyzer**. It is integrated with the **Vivado** environment, enabling design, implementation, and verification in one application.
2. **Signal Tap Logic Analyzer** - an equivalent of Chipscope, owned and maintained by Intel®. It serves for diagnostics of FPGAs supported by them [2]. Integrated with **Quartus® Prime** software [17].
3. **Reveal Analyzer** - a solution integrated with the **Lattice Diamond** platform [18] from Lattice Semiconductor. Allows for real-time debugging of most boards from this manufacturer [22].
4. Gowin Analysis Oscilloscope - a tool made available by Gowin Semiconductor. Allows for testing chips in real time. Integrated with the **Gowin EDA** environment [23].

Additionally, attention should be paid to open-source tools with a license allowing users to work on them. Such tools, by their nature, are less dependent on the company and its ecosystem. With appropriate implementation, they allow replacing software prepared by manufacturers. These tools can be divided into hardware that supports reading signals on FPGA and software that cooperates with it.

3.2.2. Programming Tools

1. **OpenOCD** - Open On-Chip Debugger is a universal tool used for establishing a connection with processors using **JTAG** and **SWD** interfaces. It allows, among other things, to program, to a limited extent, erase memory, and debug the integrated circuit. In the case of FPGA, the most important implemented application is creating a communication bridge with the reconfigurable chip and sending commands using the TCL interpreter built into this tool. The tool also allows uploading a bitstream to the FPGA [24].
2. **Sigrok** - an open-source package aiding signal analysis. Supports logic analyzers, oscilloscopes, and multimeters, from which it registers signals and helps in their analysis. To conduct graphical analysis, it uses PulseView. Ensures decoding of popular communication protocols such as I2C, SPI, UART. Can serve to capture and analyze FPGA signals for debugging [25].
3. **GTKWave** - a tool allowing for visual analysis of actions of modules written in hardware description language based on signal waveforms saved in files with formats [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

   It is a tool particularly useful for debugging problems resulting from timing dependencies and logic of cooperating modules through simulation.
4. **Communication Protocol Analyzers (USBlyzer/Wireshark)** - tools for debugging communication protocols that can be implemented, among others, in FPGA projects. **USBlyzer** [27] is intended for verifying the correctness of data transmitted via USB interface, while **Wireshark** [28] serves to capture data sent in an Ethernet network.
5. **fpgadbg2** - a tool for registering FPGA signal samples. After a trigger defined by the designer, input data is saved in FPGA memory, and then transmitted to a computer via UART interface. After processing, they are displayed using the GTKWave application [3].
6. **LiteScope** - a logic analyzer integrated into integrated circuits in LiteX SoC projects [29]. Ensures real-time monitoring of internal FPGA signals without the need for

additional external hardware and software [30]. Has a built-in wishbone-tool allowing for monitoring traffic on the wishbone bus.

7. **LiteETH** - part of the LiteX package, provides tools for debugging Ethernet interface communication in SoC projects using FPGAs from this family. Includes simulation and support for real-time debugging [31].
8. **OpenTitan Debug Infrastructure** - a verification environment for projects created in the OpenTitan environment. Offers secure debug access, JTAG integration, and support for **STM32 NUCLEO-L552ZE-Q** as a debugger, called **HyperDebug Board** by OpenTitan developers [32].

3.2.3. Hardware Tools

1. **Openbench Logic Sniffer** - a tool used for capturing and recording electrical signals. Can serve to debug FPGA and other integrated circuits. Equipped with pins and probes. Due to its high sampling rate, it is effective in observing FPGA pins. For full analysis, it needs a program like Sigrok or PulseView [33].
2. **Communication Bridges** - there are many tools allowing for the creation of a communication interface between the tested module and the test station. Such signal converters can include, among others:
   - USB-UART converter,
   - USB-SPI converter,
   - USB-I2C converter.
3. **RaspberryPI** - a single-board computer that contains controllers for typical low-level communication interfaces (e.g., I2C, SPI) and allows for efficient software implementation of the JTAG interface without the need for additional bridges. Pins exposed on the PCB allow direct connection to peripherals. For some interfaces, it may prove necessary to use a logic level converter. Additionally, some models from the RaspberryPI family have the ability to install RaspberryPI OS - a modified Linux kernel based on the Debian system. This system can serve us to install programs supporting analysis such as GTKWave or Verilator. RaspberryPI models enabling installation of RaspberryPI OS:
   - Raspberry Pi 5,
   - Raspberry Pi 4 (Model B),
   - Raspberry Pi 400,
   - Raspberry Pi 3 series *(Models B, B+, A+)*,
   - Raspberry Pi Zero series *(models Zero, Zero W, Zero 2 W)*,
   - Raspberry Pi 2,
   - Raspberry Pi 1
   - Raspberry Pi Compute Module 1,
   - Raspberry Pi Compute Module series.

3.2.4. Tools for Debugging via Circuit Simulation

Before verifying the project operation in the system, it is worth analyzing the logic of designed modules using behavioral simulation. Manufacturers such as:

- AMD,
- Intel®,
- Lattice Semiconductor,

integrate tools allowing for such analysis. These tools are:

- **Vivado Design Suite** uses the **Xylinx Simulator** tool - in short **XSIM**. Allows for simulation of hardware description languages Verilog, SystemVerilog, SystemC, and VHDL. It is maintained by Siemens AG.
- **Quartus® Prime** - uses the **ModelSim** tool, adapted for work with Intel® FPGAs. This tool is maintained by Siemens AG.

  Lattice Diamond - uses **ModelSim** and **Active-HDL** tools, maintained by Aldec.

There are also independent solutions on the market, including:

1. **QuestaSim** - a tool maintained by Siemens AG. Unlike **ModelSim**, it serves to simulate projects composed of many millions of gates. They can be described in SystemVerilog and VHDL languages [34].
2. **Incisive® Enterprise Simulator** - a solution supporting simulation of modules described in languages *e, Verilog, VHDL, SystemC, SystemVerilog, CPF, PSL, OVL, SVA, Simulation* [35]. Created and maintained by Cadence.
3. **VCS** - a tool maintained by Synopsis. Enables simulation of Verilog, VHDL, and SystemVerilog 3.1a languages [36].
4. **Cocotb** - a Python library for writing test environments based on cooperation with integrated programs used in FPGA and ASIC projects. Enables testing protocols, co-simulation using simulators such as ModelSim, Icarus Verilog, and Verilator, and verification of FPGA logic [37].
5. **Verilator** - an open-source Verilog simulator and compiler. Converts Verilog language to C++ for high-speed simulation. Serves for cycle-accurate timing analysis and extensive testing in FPGA simulations. Allows for verification of modules written in hardware language based on simulation [38].
6. **SymbiFlow** - a collection of open-source FPGA tools supporting many FPGA chip families. Integrates open-source tools and those closed to external user interference. For debugging, it uses an open tool - **VPR**, which allows for examining the schematic and setting traps in the verified architecture schematic. The whole operates based on simulation, not on a physical chip. [39][40].
7. **IceStorm** - part of the open-source toolchain for FPGA chips supported by YosysHQ. Its main application is documentation and bitstream creation for Lattice iCE40 reconfigurable chips. It allows inspecting resources and ensures a tool for debugging

timing dependencies based on the bitstream file, using the integrated tool *icetime* [41].

3.3. Problems with Popular FPGA Diagnostics Systems

Taking into account all tools available on the market, there is no single open-nature solution that would allow for simple and universal real-time debugging of FPGAs with differing architectures. In the case of running systems employing different, specialized FPGAs, such a universal tool would facilitate the work of an engineer looking for a bug on a given chip. Additionally, such a tool should not be difficult to implement, and consequently hinder compilation. It should not introduce errors in the operation of described modules after implementation and should not consume many FPGA resources. Using tools available from manufacturers forces a choice between using the full functionality of the FPGA or debugging the system. One of the problems may be that the SoC chosen by us does not support two JTAG connections using one connector, which prevents communication with a PC to send data via JTAG and simultaneous viewing of signals. If our system requires sending data via JTAG, the debugging process can be hindered. Tools available on the market that feature portability are costly, hard to access, or limited in their capabilities. Whereas the more advanced ones are closed in their ecosystems and their portability is limited to one manufacturer. Using many of them simultaneously can be inconvenient. The **Portable system for diagnostics and start-up of digital systems implemented in FPGA systems** aims to address these problems, ensuring users a convenient, easy-to-use, and universal system for debugging FPGA chips, while simultaneously limiting its costs to a minimum.

4. Designing a Portable Diagnostics System

4.1. Design Requirements

The **Portable system for diagnostics and start-up of digital systems implemented in FPGA systems** must meet basic assumptions:

1. Be **portable**, i.e., work on the largest possible number of FPGA platforms, produced by different manufacturers, possessing different architectures and different environments for project debugging.
2. Enable **diagnostics** and **start-up** of digital systems after synthesis and implementation. In some projects, during start-up, errors appear that were not visible during system simulation due to the influence of external factors, potential path length, or interference caused by external electronic circuits. This system is intended to enable pinpointing and eliminating such errors.

Additionally, when designing such a system, its usability and convenience for the end user should be taken into account. Taking these facts into consideration, additional assumptions of this project are:

1. Comfort of portability - this system should be functionally the same, or significantly similar for every type of FPGA possible to debug. The possibility of development, including adding new FPGAs, modifying connections and used pins should be relatively simple, so that it is easy to transfer the system between projects.
2. Availability - materials, i.e., software, along with hardware needed to run the diagnostics system should be easily available and well documented.

4.2. Selection of Hardware Platform and Applied Technologies

Due to the above assumptions, I decided to develop the already existing solution fpgadbg2 [3]. The hardware description language I decided on is VHDL. To make the project easily available, I decided to additionally use the Raspberry Pi platform [42], which serves in this project as an analyzer of recorded signals and a personal computer for presenting waveforms on the FPGA. The signal converter is created in Python, using C libraries made available by GTKWave, serving to create a *.lxt* file, enabling sample display in the GTKWave browser [43].

4.3. System Design Description

The project consists of three main parts, specifically:

- Signal recorder, interchangeably called **fpgadbg core** hereinafter. Its task is to save a specific amount of samples in FPGA memory after an event defined by the user.
- Data analyzer, interchangeably called **fpgadbg conv** hereinafter. Its task is to send configuration bytes to the FPGA, and then receive the signal with data from it. After saving and processing them, they should be processed to a file in *.lxt* format. Finally, it should perform data presentation using GTKWave.
- Communication interface, interchangeably called **wrapper** hereinafter. Its task is to ensure connection between the started project and the external converter. Due to the hardware capabilities of RaspberryPI, direct connections will be made between RaspberryPI pins and the tested SoC containing the FPGA using wires.

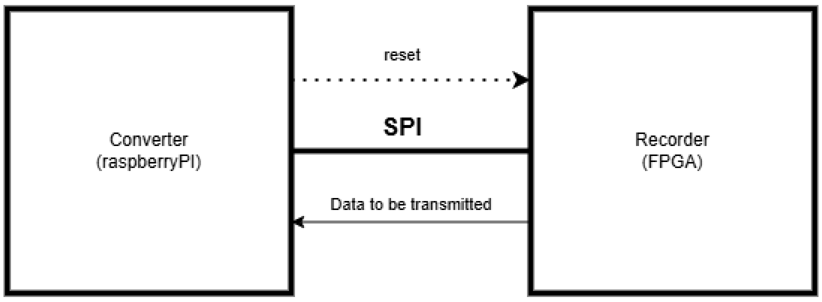

Figure 4.1. Simplified scheme of the portable system for diagnostics and start-up of digital systems implemented in FPGA systems project.

5. System Implementation

5.1. Component Parts

5.1.1. Signal Recorder

The first part of the project is a module allowing for real-time observation of signals on the FPGA and their recording to memory. The memory in which recorded samples are to be stored should be easily accessible on every FPGA where the **Portable system for diagnostics and start-up of digital systems implemented in FPGA systems** is operating. They should not require sequential initialization to enable the tool to register signals in every, including the initial, phase of the validated project's operation. This system should not rely on external peripherals to facilitate its integration into various systems utilizing FPGA. Reconfigurable chips implement BRAM memory blocks, in technical documentation interchangeably called *EBR* [44], which can be used as memory for saving samples. Depending on the FPGA architecture, these blocks typically have a size of:

- 18Kb or 36Kb for reconfigurable chips made in the architecture used by AMD, and formerly Xylinx;
- 640b, 9Kb, 20Kb, 144Kb for FPGAs created in the architecture used by Intel®.

Implementation of BRAM memory blocks in FPGA chips using hardware description languages is a thoroughly documented issue. There are many solutions on the network implementing various types of BRAM blocks, differing in resource occupancy and possible operations. Many of them are available under an open license. In the built tool, implementation of DPRAM is necessary, which will enable saving sampled signals to memory in real time during signal testing at a specific trigger moment and subsequent reading to transmit data to a peripheral device to enable their further analysis. In the construction of the signal recorder, an implementation created in the fpgadbg2 tool [3] was used due to the module integrated with it sending basic data about the instantiated memory to external interfaces. After minor changes, as an additional way of verifying the correctness of work with the tool, it also transmits feedback to the user about the number of declared signals. This module implements TDPRAM memory, which allows simultaneous writing and reading of data.

There exists a possibility of modifying the volume of used resources, increasing or decreasing the number of samples that are recorded by default, as well as the length of the tested input signal whose samples will be saved. This involves modification of the declaration of the aforementioned information before tool synthesis, and consequently modification of the number of instantiated memory cells. The placement of used resources on the FPGA, managed by tools used in the process, takes place during project implementation on the FPGA.

This part of the project was made in VHDL hardware description language; it is implemented directly in the FPGA. Due to differences in FPGAs from various distributors,

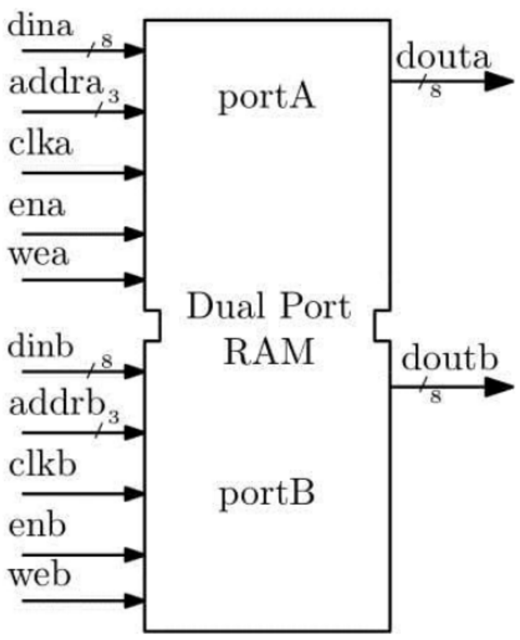

Figure 5.1. TDPRAM cell diagram [45].

resulting from architecture and text requirements of their compilers as well as differences between FPGAs of the same distributor, resulting from different pinout and use of a different oscillator, modification of the main module of the system for diagnostics and start-up of digital systems will be needed. Example projects have been prepared that can help in better understanding and implementing the tool.

5.1.2. Data Analyzer

To enable validation of the operation of the tested project implemented on FPGA, data saved in the FPGA must be downloaded and presented to the user. To create the module enabling this, RaspberryPI 5 was used. The RaspberryPI OS operating system has support for most standard programs, libraries, and drivers available for the Debian system, on which it is based. Exposed hardware pins allow for direct connection with the debugged reconfigurable chip. For communication between RaspberryPI and FPGA modules, the SPI interface was used. The driver, adapted to work with the **signal recorder**, was written in Python, using the SPIDEV library [46]. Presentation of data saved in BRAM memory in the form of a "waveform" is performed using the GTKWave application. Tony Bybell provides, under an open license, the `lxt_write` library [43], which allows creating a file in `.lxt` format, supported by this program. Use of this library and necessary operations of analyzing the bitstream received from the reconfigurable chip by RaspberryPI before conversion were performed in Python.

The single-board computer is responsible for:

1. Configuring the Signal Recorder.

2. Selecting the tested FPGA using the CS line.
3. Receiving data with waveform recording, sent by the FPGA.
4. Preparing data for conversion.
5. Checking the correctness of received data.
6. Converting data to *.lxt* format.
7. Displaying data in GTKWave.

To prepare libraries and install required software, an executable file *install.sh* was prepared. Execute permissions should be granted to it, and then it should be run. System occupancy after installation is 7.5GB. SPI inside the data converter is adapted to work with the Signal Recorder. *In case of use in other projects, the correctness of data reception should be verified.* Tests were performed on RaspberryPI 5, however due to backward compatibility of used libraries, use of any product from the Raspberry family should be possible without tool modification. This allows for reducing the initial price of the Portable system for diagnostics and start-up of digital systems implemented in FPGA systems.

In case of inability to connect a screen for displaying results, there is a possibility to download the file using the SSH protocol and then present it on a computer chosen by the user. The file from the last validation of the reconfigurable chip is not deleted. Use of hardware other than RaspberryPI (e.g., a computer with installed system based on Debian OS together with a USB-SPI communication bridge) is possible, due to standard libraries used in the project as communication interface drivers. However, this may require tool modification.

5.1.3. Communication Protocol Description

The communication protocol used between RaspberryPI and the part recording samples on FPGA is SPI. It is an interface characterized by greater stability than asynchronous interfaces such as UART and, due to the use of a larger number of communication lines, smaller errors between different chips without common ground space.

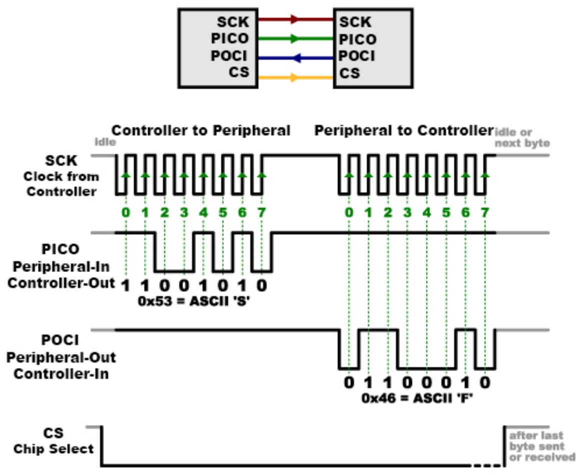

Figure 5.2. Diagram of standard SPI communication [47].

In the case of the **Portable system for diagnostics and start-up of digital systems implemented in FPGA systems**, RaspberryPI is the master, and FPGA is the slave device. Due to hardware limitations of RaspberryPI, the theoretical maximum operating frequency of the interface is 125 MHz, however it is recommended to limit it to:

– 50 MHz for short and good connections,
– less than 25 MHz for long wires.

RaspberryPI allows connecting two slave devices to one SPI interface, and consequently, our system allows debugging two FPGAs simultaneously. *By default, SPI0 CS0 are used. It is possible to use other communication lines, however small modifications in the file `main_fpgadbg_SPI.py` will be needed.*

SPI communication declaration in code:

```
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
```

In case of desire to use other communication lines, values `SPI_bus` and `SPI_chip_select` should be set accordingly. When using SPI 1 bus, other pins must be used for connection with FPGA:

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

5.2. Operation Process

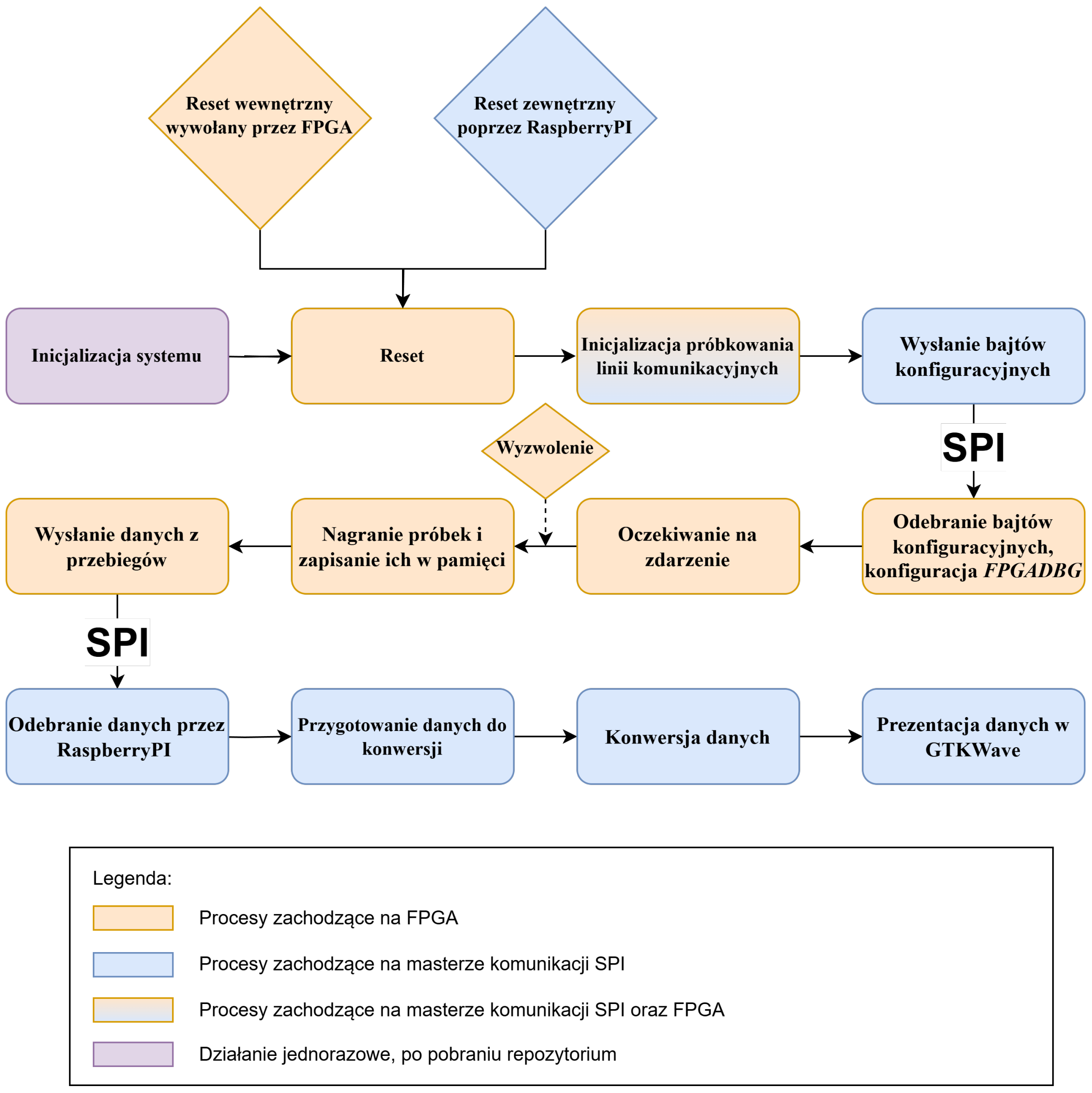
Reset wewnętrzny wywołany przez FPGA
Reset zewnętrzny poprzez RaspberryPI
Inicjalizacja systemu
Reset
Inicjalizacja próbkowania linii komunikacyjnych
Wysłanie bajtów konfiguracyjnych
SPI
Wyzwolenie
Wysłanie danych z przebiegów
Nagranie próbek i zapisanie ich w pamięci
Oczekiwanie na zdarzenie
Odebranie bajtów konfiguracyjnych, konfiguracja *FPGADBG*
SPI
Odebranie danych przez RaspberryPI
Przygotowanie danych do konwersji
Konwersja danych
Prezentacja danych w GTKWave
Legenda:
Procesy zachodzące na FPGA
Procesy zachodzące na masterze komunikacji SPI
Procesy zachodzące na masterze komunikacji SPI oraz FPGA
Działanie jednorazowe, po pobraniu repozytorium

Figure 5.3. Block diagram of chip operation.

1. **System Initialization** - installing required libraries and software. Then compiling them appropriately or adding to system paths. *This step needs to be performed only once - before the first run of the tool. Then it should be skipped.*
2. **Reset** - restoring default states of individual FPGA modules and their variables to default values. Reset is triggered by GPIO state read. Input must be defined in the constrain file. Depending on project need, two options for connecting input have been made available:
   - **Internal reset triggered by SoC** - if the chip enables reset using a button, switch, or other easily accessible peripheral.

  - **External reset via RaspberryPI** - reset issued using RaspberryPI (*default set to GPIO 3 of RaspberryPI*).

3. **Initialization of communication line sampling** - *MOSI* and *SPIclock* lines between master and slave module are sampled using a 2-bit sampler, using majority decision. This allows for more accurate sampling of communication lines eliminating potential errors resulting from single bit value changes in the signal resulting from noise. The sampler is enabled using CE. In case of high state, the sampler always returns value 1 and reading of configuration data nor SPI clock takes place. This allows connecting several FPGA chips parallel to each other. Using the sampler forces a maximum *SPIclock* clock value:

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

4. **Sending configuration bytes** - RaspberryPI sends 3 configuration bytes to the FPGA chip, which define the number of samples saved in memory after user-defined trigger and selected configuration, which allows defining several events, or different recorded signals without the need for re-synthesis, implementation, and bitstream upload.
5. **Receiving configuration bytes, *FPGADBG* configuration** - configuration bytes are described in the following way:

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]

Table 5.1. Meaning of bits in configuration bytes

Legend:

- **b[n]** - byte sent via SPI from master.
- **t[n]** - 16-bit variable defining the amount of samples to be saved. Maximum value is 65536.
- **x[n]** - 3 bits enabling debugging of 8 different parts of the project, by dependency of saved signals on variable value as well as dependency of starting waveform recording on different triggering events without need for re-synthesis, implementation, and uploading bitstream to FPGA.

6. **Waiting for event** - *trigger should be invoked by user after condition met, e.g., when tested variable reached specific value.*
7. **Recording samples and saving them in memory** - after event, samples are saved in BRAM memory of TDPRAM type.
8. **Sending waveform data** - firstly bytes with encoded information about the waveform are sent:

A. `log2samples` - byte informing about maximum number of samples that could have been recorded.
B. `width` - byte informing about total bit length of recorded signals.
C. `number_of_signals` - byte informing about declared number of tested signals.
D. `trigger_position` - 1 or 2 bytes informing about trigger moment. Transmitted LSB first.
E. `stop_position` - 1 or 2 bytes informing about moment of writing last bit to buffer. Transmitted LSB first.

Then recorded waveforms are sent in number defined by previously sent configuration bytes using LSB first method.

9. **Receiving data by RaspberryPI** - Master receives data and saves in a tuple table.
10. **Preparing data for conversion** - Removal from table of unnecessary:
    - **initial noise,**
    - **bits, read from line during reading data from memory.**

    Verification of correctness of received data occurs and saving them as bitstream.
11. **Data conversion** - data are decoded, values log2samples, width, trigger position, and stop position are decrypted. Then a *.lxt* file is generated containing read data. It serves for data presentation using the GTKWave program.
12. **Data presentation in GTKWave** - data are ready for debugging, their preview is made available for the user using the GTKWave application.

6. Diagnostics of Selected Digital Systems

6.1. FPGA Chips Used for Test Implementation

Implementation of the **FPGADBG** system was performed on three different systems:

Table 6.1. SoC systems on which the project was implemented.

Development Board	FPGA	Manufacturer
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. Functional Tests

6.2.1. Integration Tests

Separate integration tests of fpgadbg core and fpgadbg conv parts were performed. Both tests were conducted by simulating them using data that could correspond to the actual operation of the program.

```
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. Simulation file used for testing fpgadbg core operation.

After correctly decoding metadata [Listing 3], the program creates a *.lxt* file, which will display the constant value of the tested signal [Listing 2].

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

data = "1000101000100001011000010000000101100011000000011011111000000000100000011000000000000000110111110000000001000000110000000000000001"
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. Simulation file for testing fpgadbg conv.

```python
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. Code fragment serving to decode metadata.

6.2.2. System Tests

Operation of the entire tool as a whole was tested on input data algorithms as tested signals:

- constant number,
- counter,
- blinking.

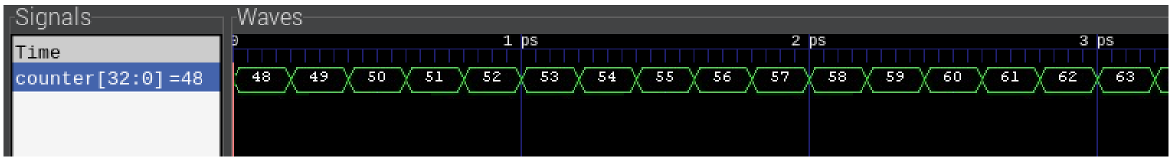

Figure 6.1. Registered counter waveform.

6.2.3. Usability Tests

Usability test was performed by integrating the **Portable system for diagnostics and start-up of digital systems implemented in FPGA systems** into the OLED display interaction project on PmodOLED [53][54] module from Digilent. The PYNQ-Z2 development board [48] was used for this.

6.3. Research on System Resource Usage

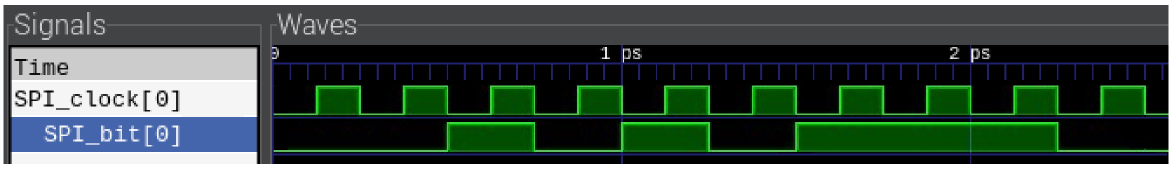

Figure 6.2. Registered waveforms of internal SPI clock and SPI bit sent to buffer as part of screen initialization. Visible command is 0xAE.

Table 6.2. Comparison of FPGA resource usage in the OLED display interaction project on Pmod-OLED [53][54] module from Digilent attached to PYNQ-Z2 development board [48] with 10-bit input signal width and 65536 recorded samples.

Resource	No fpgadbg	Implemented 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
Total power consumption [mW]	99	108

Table 6.3. Comparison of resource usage for selected number of samples in case of testing input signals with total width of 33 bits.

Resource** / FPGA	1024			16384			65536		
	LUT	FF	Power	LUT	FF	Power	LUT	FF	Power
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	*

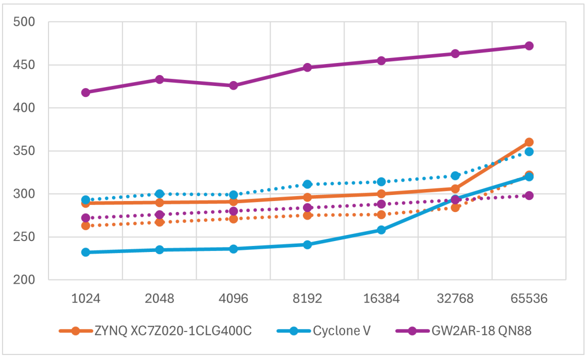

Figure 6.3. Graph of resource usage by Portable system for diagnostics and start-up of digital systems implemented in FPGA systems in case of testing input signals with total width of 33 bits. Dashed lines mark FF usage, and solid lines LUT usage.

Table 6.4. Comparison of resource usage for selected number of samples in case of testing input signals with total width of 8 bits.

Resource** / FPGA	**1024**			**16384**			**65536**		
	LUT	FF	Power	LUT	FF	Power	LUT	FF	Power
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	*

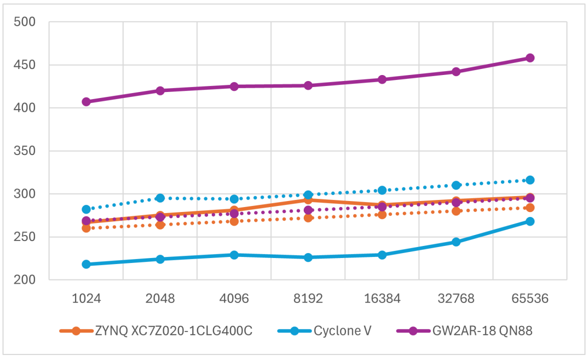

Figure 6.4. Graph of resource usage by Portable system for diagnostics and start-up of digital systems implemented in FPGA systems in case of testing input signals with total width of 8 bits. Dashed lines mark FF usage, and solid lines LUT usage.

* Quartus Lite does not provide power usage analysis capability [55]. Due to Pro version price, I decided not to perform these tests.

** Power is given in milliwatts with 1 milliwatt accuracy.

Tests showed that not every FPGA deals equally well with optimizing resource usage when using the portable system for diagnostics and start-up of digital systems implemented in FPGA systems. The gowin EDA environment [19], optimizing the project for GW2AR-18 QN88 FPGA [19], performed the worst. This was a predictable result, due to much higher prices, and consequently product quality of competitors. ZYNQ XC7Z020-1CLG400C[49] and Cyclone V[52] achieved similar results. It is worth noting that LUT and FF FPGA usage, with a significant increase in the number of tested samples, does not increase drastically. It is worth considering using the largest possible amount of recorded samples to see the largest possible part of the tested signal.

7. Summary and Conclusions

7.1. Thesis Summary

The Portable system for diagnostics and start-up of digital systems implemented in FPGA systems is a universal tool, thanks to which, after a short introduction, it is easy to debug various FPGA systems, regardless of manufacturer architecture. It can be used as an open and available under free license IP core allowing for FPGA debugging.

7.2. Final Conclusions

Tests showed that not every FPGA deals equally well with optimizing resource usage when using the portable system for diagnostics and start-up of digital systems implemented in FPGA systems. When using the project and further developing it, manufacturers' limitations should be taken into account. In case of joining it to a project, one should explore if it will surely fit in FPGA resources. It does not occupy much space, however, when approaching maximum resource utilization by connecting with other modules, optimization may be worse. Additionally, possible complications, less frequently encountered in case of debugging using traditional methods, should be considered. such complications can include, among others:

- not working pin on RaspberryPI,
- broken wire between master and slave,
- lack of internet (in case of using SSH).

This does not change the fact that it is an easily portable and universal tool. It is worth considering its application if we operate on many different FPGAs ensuring different programming environments.

7.3. Proposals for Further Development

Development of the portable system for diagnostics and start-up of digital systems implemented in FPGA systems is possible in many different directions:

- **Support for additional FPGAs** - Adding additional FPGAs, using existing templates will increase the universality of the solution.
- **Adding additional communication interface** - wrapper using JTAG, or SWD could ensure greater possibilities. It is worth considering using OpenOCD.
- **Using cloud** - data downloaded from FPGA could be immediately shared in the cloud with the possibility of later download.
- **Adding synthesis capability at RaspberryPI level** - Using other open-source solutions, one could enable the user to synthesize, implement, and upload bitstream to selected FPGA architectures.
- **Passing information about debugged signals directly from FPGA** - currently, the user must declare variables they want to debug in the main VHDL module, and then

do it again in python files. This process should be simplified to exclude possible mistakes.

- **Examining power consumption on Cyclone V FPGA using Quartus Pro**

Source code has been made available under GNU General Public License, *Free Software Foundation.*

References

[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”, Accessed (25.11.2024): `https://koral.ise.pw.edu.pl/~wzab/fpgadbg/`, 2006.

[4] Markets and 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”, Accessed (25.11.2024): `https://www.marketsandmarkets.com/Market-Reports/fpga-market-194123367.html`, 2024.

[5] hardwarebee, “List of fpga companies”, Accessed (26.11.2024): `https://hardwarebee.com/list-fpga-companies/`, 2018.

[6] J. Schneider and I. Smalley, “What is a field programmable gate array (fpga)?”, Accessed (25.11.2024): `https://www.ibm.com/think/topics/field-programmable-gate-arrays`, 2024.

[7] C. work edited by Grzegorz Karpiel, *FPGA chips in control and regulation systems.* AGH University of Science and Technology in Krakow: Department of Robotics and Mechatronics, 2023.

[8] N. Instruments®, “Fpga fundamentals: Basics of field-programmable gate arrays”, Accessed (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”, Accessed (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)”, Accessed (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”, in *Innovations and technologies in engineering*, Eğitim Yayınevi, 2022, pp. 171–183.

[12] K. Sungho, “High-speed incoming infrared target detection by fusion of spatial and temporal detectors”, *Frontiers in Infrared Photodetection*, vol. EISSN 1424-8220, no. 15(4), pp. 7267–7293, 2015.

[13] AMD, *Aerospace defense solutions, space*, Accessed (26.11.2024): `https://www.amd.com/en/solutions/aerospace-and-defense/space.html`, 2024.

[14] T. Wu, W. Liu, and Y. Jino, “An end-to-end solution to autonomous driving based on xilinx fpga”, in *2019 International Conference on Field-Programmable Technology (ICFPT)*, IEEE, 2019.

[15] 3BL, “Road experience management targets accuracy to enable fully autonomous driving”, Accessed (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”, Accessed (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, *Chenxi family*, Accessed (19.01.2024): `https://www.gowinsemi.com.cn/prod_view.aspx?TypeId=10&FId=t3:10:3&Id=167`, 2024.

[20] *Dictionary of polish language - definition of word 'diagnostics'*, Accessed (14.03.2024): `https://sjp.pwn.pl/slowniki/diagnostyka`, 2024.

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

[22] *Reveal user guide*, Accessed (11.01.2025): `https://www.latticesemi.com/~/media/328D471BF2C74EB1907832FAA6FB344B.ashx`, Lattice Semiconductors, 2015.

[23] *Gowin analyzer oscilloscope user guide*, Accessed (11.01.2025): `https://www.gowinsemi.com/upload/database_doc/37/document/5bfcfed078251.pdf`, Gowin Semiconductor, 2018.

[24] D. Rath, *Openoc*, Accessed (11.01.2025): `https://openocd.org/doc-release/html/index.html`, 2024.

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

[26] U. Finkelstein, *Gtkwave 3.3 wave analyzer user's guide*, Accessed (12.01.2025): `https://gtkwave.sourceforge.net/gtkwave.pd`, 2020.

[27] F. U. Analyzer, *Hdd software*, Accessed (11.01.2025): `https://freeusbanalyzer.com/`, 2024.

[28] R. Sharpe, *Wireshark user's guide*, Accessed (11.01.2025): `https://www.wireshark.org/docs/wsug_html_chunked/`, 2024.

[29] E. Digital, *Litex*, Accessed (11.01.2025): `https://github.com/litex-hub`, 2025.

[30] E. Digital, *Litescope*, Accessed (11.01.2025): `https://github.com/enjoy-digital/litescope`, 2025.

[31] E. Digital, *Liteeth*, Accessed (11.01.2025): `https://github.com/enjoy-digital/liteeth`, 2025.

[32] *Opentitan project*, lowRISC C.I.C., 2024.

[33] W. Labs, *Openbench logic sniffer*, Accessed (12.01.2025): `http://dangerousprototypes.com/docs/Open_Bench_Logic_Sniffer`, 2017.

[34] *Questa® sim user's manual*, Accessed (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*, Accessed (24.01.2025): `http://pdf2.solecsy.com/565/c02013ae-ffce-4436-9866-182120ab263c.pdf`, Cadence®, 2011.

[36] *Vcs®/vcsi™ user guide*, Accessed (24.01.2025): `https://users.ece.utexas.edu/~patt/10s.382N/handouts/vcs.pdf`, Synopsys, 2008.

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

[38] W. Snyder, *Verilator user's guide*, Accessed (12.01.2025): `https://verilator.org/guide/latest/`, 2024.

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

[40] K. E. Murray, T. R. Ansell, K. Rothman, and A. Comodi, "Symbiflow vpr: An open-source design flow for commercial and novel fpgas", Accessed (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*, Accessed (11.01.2025): `https://github.com/YosysHQ/icestorm`, 2024.

[42] RaspberryPI, *Raspberry pi documentation*, Accessed (12.01.2025): `https://www.raspberrypi.com/documentation/`, 2025.

[43] T. Bybell, *Lxt*$_w rite$, Accessed (14.01.2024): `https://github.com/Parrot-Developers/lttng2lxt/tree/master`, 2005.

[44] Nandland, "What is a block ram (bram) in an fpga?", Accessed (25.11.2024): `https://nandland.com/lesson-15-what-is-a-block-ram-bram/`, 2022.

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

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

[47] M. Grusin, "Serial peripheral interface (spi)", Accessed (12.01.2025): `https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all`, 2018.

[48] *Pynq-z2 reference manual v1.0*, Accessed (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*, Accessed (19.01.2025): `https://docs.amd.com/v/u/en-US/ds190-Zynq-7000-Overview`, AMD, 2018.

[50] Sispeed, *Tang nano 20k*, Accessed (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*, Accessed (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*, Accessed (19.01.2024): `https://www.intel.com/programmable/technical-pdfs/683801.pdf`, Intel®, 2023.

[53] *Pmodoled™ reference manual*, Accessed (11.01.2025): `https://digilent.com/reference/_media/reference/pmod/pmodoled/pmodoled_rm.pdf`, Diligent®, 2016.

[54] *Digilent pmod™ interface specification 1.2.0*, Accessed (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*, Accessed (14.03.2024): `https://www.intel.co.kr/content/dam/www/central-libraries/us/en/documents/quartus-prime-compare-editions-guide.pdf?`, 2024.

List of Symbols and Abbreviations

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
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 – Ultrasonography
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

List of Figures

List of Tables

List of Listings

List of Appendices

Appendix 1. Github Repository

`https://github.com/PawelMurdzek/fpgadb3`