VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Eetu Sippola

VENDOR INDEPENDENT FPGA DEVELOPMENT

Technology and Communication
2023

VAASAN AMMATTIKORKEAKOULU
Tietotekniikka

TIIVISTELMÄ

Tekijä	Eetu Sippola
Opinnäytetyön nimi	Toimittajasta riippumaton FPGA-kehitys
Vuosi	2023
Kieli	englanti
Sivumäärä	51
Ohjaaja	Santiago Chavez Vega (VAMK), Petri Ylirinne (Danfoss Drives)

Tämä opinnäytetyö tehtiin Danfoss Oy:lle. Opinnäytetyön tavoitteena oli tutkia toimittajasta riippumattoman FPGA-kehityksen mahdollisuuksia ja antaa näkemys FPGA-kehityksen ekosysteemin kypsyydestä toimittajasta riippumattomien menetelmien ympärillä. Toimittajasta riippumattomat menetelmät ovat tavanomaisten sulautettujen järjestelmien ohjelmistokehityksen peruselementtejä, mutta FPGA:n saralla nämä ovat vasta tulossa.

Tässä opinnäytetyössä tehtiin yksinkertainen toimittajariippumaton toteutus käyttäen vain geneeristä HDL:ää ja toimittajasta riippumatonta prosessoria avoimella käskykanta-arkkitehtuurilla. Tämä toteutus käännettiin sitten onnistuneesti muutaman eri valmistajan FPGA-piireille käyttämällä EDA-työkalua. Tätä prosessia käytettiin referenssinä kartoittaessa toimittajariippumattomia menetelmiä ja niiden mahdollisuuksia FPGA-kehityksessä. Myös muita lähteitä ja menetelmiä tutkittiin pääasiassa käännösprosessia varten. Tietoa kerättiin itse prosessista, tieteellisistä artikkeleista, julkaisuista sekä valmistajan käsikirjoista ja dokumentaatiosta.

Tutkimuksen aikana todettiin, että tämän opinnäytetyön menetelmillä on mahdollista toteuttaa ainakin jossain määrin tai jopa kokonaan toimittajariippumattomia projekteja. Toimittajasta riippumattomien menetelmien käyttäminen tuottaa lisätyötä varsinkin, jos projekti on alun perin kehitetty sisältäen toimittajariippuvaisia osia.

Tarve vaihtoehtoiselle käännösprosessille havaittiin, koska toimittajan tarjoaman EDA-työkalun käyttö vähentää projektin toimittajan riippumattomuutta. Hyväksi vaihtoehtoiseksi menetelmäksi todettiin toimittajakohtaisten työkalujen skriptaaminen kehittäjien itsensä toimesta. Kaiken kaikkiaan todettiin, että tässä opinnäytetyössä kuvatuilla menetelmillä on mahdollista pyrkiä toimittajariippumattomuuteen.

Avainsanat FPGA, toimittajariippumaton, RISC-V

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietotekniikka

ABSTRACT

Author	Eetu Sippola
Title	Vendor Independent FPGA Development
Year	2023
Language	English
Pages	51
Name of Supervisor	Santiago Chavez Vega (VAMK), Petri Ylirinne (Danfoss Drives)

This thesis was done for Danfoss Oy. The objective of the thesis was to research the possibilities of vendor independent FPGA development and offer insight to the maturity of the ecosystem around vendor independent methods in FPGA development. Vendor independent methods are a staple in conventional software development in embedded systems but in the field of FPGA these methods are just emerging.

In this thesis a simple vendor independent implementation was done using only generic HDL and vendor independent processor with open instruction set architecture. This implementation was then successfully translated to a couple of different devices using an EDA tool. This process was used as a reference for mapping vendor independent methods and their possibilities in FPGA development. Also, other sources and methods were investigated mainly for the translation process. Information was gathered from the process itself, scientific articles, publications and manufacturer manuals and documentation.

During the research it was concluded that using the methods in this thesis it is fully or at least to some extent possible to have vendor independent projects. Using vendor independent methods will produce some extra work especially if the project is initially developed having vendor dependent parts.

A need for an alternative process for translating was also discovered since using a vendor provided EDA tool will lessen the vendor independence of a project. For an alternative method scripting the vendor specific tools by the developers themselves was deemed plausible. All in all, it was concluded that it is feasible to strive for vendor independence using the methods described in this thesis.

Keywords *FPGA, vendor independent, and RISC-V*

CONTENTS

LIST OF FIGURES AND TABLES

LIST OF ABBREVIATIONS

AMBA	Advanced microcontroller bus architecture
APB	Advanced peripheral bus
BSCANE2	Boundary-scan user instruction
CLB	Configurable logic block
CPU	Central processing unit
DFF	D Flip-Flop
EDF	Electronic design format
FPGA	Field programmable gate array
GDB	GNU Project Debugger
GP	General purpose
HDL	Hardware description language
IDE	Integrated development environment
IP	Intellectual property
ISA	Instruction set architecture
JTAG	Joint test action group
LED	Light-emitting diode
MIPS	Million instructions per second
IO	Input/output

PLL	Phase locked loop
RISC	Reduced instruction set computer
SBT	Scala build tool
SDC	Synopsys design constraints
SoC	System on chip
TAP	Test access port
RAM	Random-access memory
TCL	Tool command language
UART	Universal asynchronous receiver transmitter
VHDL	Very high speed integrated circuit HDL
VQM	Verilog Quartus mapping
XDC	Xilinx design constraints

1 INTRODUCTION

1.1 Background Information

Vendor independent methods are a staple in typical software designing, where different IDEs (Integrated Development Environment) and tools are almost always fully vendor independent and easily accessible. In FPGA design however this trend is just emerging. Many companies have been announcing their support for vendor independent designs in recent years using the RISC-V ISA (Reduced instruction set computer and version number 5 (V), which is an open-source ISA (Instruction Set Architecture). Further information is given in Chapter 2.2). [1] [2]

Just about all major FPGA (Field Programmable Gate Array) vendors already have an official RISC-V support (exception being Xilinx (co-branded as AMD Xilinx after stock-swap deal in February of 2022) [3]). In 2019 Lattice and SiFive (RISC-V based processor vendor) announced a collaboration on delivering new optimized processor cores based on RISC-V ISA [4], also RISC-V International announced in 2022 that Intel has joined the organization and will be supporting open RISC-V architecture [5], and in the case of Xilinx, there are options for RISC-V processors that have been built to Xilinx devices, for example, the processor used in this thesis (VexRiscv). It is important to note that even though many major FPGA vendors have shown their support for RISC-V their RISC-V implementing devices are proprietary products and therefore, not open-source.

This trend is intriguing to observe and proposes a question about vendor independent methods also in the very hardware centric field of FPGA development.

---

[1] Arstechnica. 2023.. Accessed 22.03.2023.
[2] RISC-V. 2021. Accessed 22.03.2023.
[3] AMD. 2022. Accessed 22.03.2023.
[4] Lattice. 2019. Accessed 22.03.2023.
[5] Intc. 2022. Accessed 22.03.2023.

With vendor independent methods and using RISC-V softcore CPU (Central Processing Unit) it would be possible to have device independent designs that would hypothetically be easy to port to different FPGA vendors devices.

If applicable, these methods would greatly ease both procurement and license processes which both are a typical inconvenience in manufacturing products that use some sort of processors. This would mean a negotiation leverage when negotiating prices and deals with vendors since the company can use any type of devices without the need to think about proprietary pieces of software or tools that require the usage of proprietary components.

1.2 Objective

The objective of this thesis is to research vendor independent FPGA development and its capabilities in company setting. The objective is to create an FPGA design that is as vendor independent as possible, using generic HDL (VHDL, Verilog and/or SystemVerilog). Using generic HDL (Hardware Description Language, typically FPGA development is done using some HDL for the circuit presentation) in this context means writing a hardware description as vendor independently as possible, so that no vendor specific pieces of logic are implemented. Then the design is implemented on different vendors chips using a multi-vendor synthesis tool. Thus, the same implementation could be implemented for the circuit of any or at least as many circuit manufacturers as possible.

The thesis consists of two main stages. The first is to create the implementation as vendor independently as possible. This implementation is done using generic HDL and an FPGA-compatible open-source RISC-V ISA based processor Vex-RiscV.

The second main stage is to investigate possible tools that could be used to translate this model to the devices of different vendors. The purpose is also to investigate how easily the implementation could be translated using these possible tools

and whether it requires, for example, some vendor specific adaptation packages to function smoothly.

The purpose of this thesis is to find out the effects of vendor independent methods in FPGA development. The thesis also discusses the challenges encountered and how they can be avoided.

1.3 Structure of the Thesis

The second chapter will go through all the relevant tools and technologies and their basic information. After that in the third chapter the designing of the implementation to be used as a test piece in the translating process is described and the steps taken to implement this design and running/debugging software on that design. Then in the fourth and fifth chapters the translation to different vendors devices is described and the results of said translation process. Lastly some conclusions are drawn on what these results mean for the future and how this topic can be researched even further, also some insights on how this affects FPGA development now or in the future are given.

The process will have the workflow that is represented in Figure 1. First the design files are developed with general HDL programming vendor as independently as possible then the work transfers to an EDA tool for translating the implementation for different vendors through steps 2-4. The work in the EDA tool reminds a typical FPGA development going from compilation to place and route which will ultimately give us the needed output files for programming the target device.

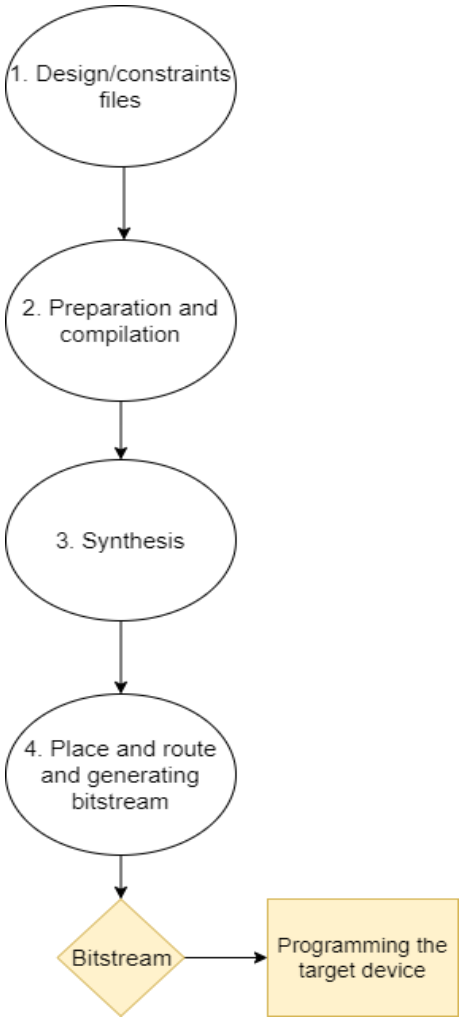

Figure 1. Workflow.

1.3 Danfoss

Danfoss is a Danish company founded in 1933. They have a vast expertise in many fields including but not limited to power solutions, drives, and climate solutions. With 42,000 employees worldwide and quality, reliability, and innovation in the epicenter of their work, Danfoss is well on the way towards carbon-neutral global operations by 2030 and a leading technology partner for their customers who

want to decarbonize through energy efficiency, machine productivity, low emissions, and electrification. [6]

Figure 2 Danfoss logo [6].

The Paris agreement (a legally binding treaty on climate change [7]) has set in motion a global motivation for green and sustainable energy and Danfoss is in the forefront of this motion since their products focus on getting the most out of the existing systems and therefore saving a lot of energy in industry and transportation.

In Finland Danfoss Drives segment focuses on manufacturing and developing frequency converters. Danfoss Drives is a global leader in AC/DC and DC/DC power conversion, as well as variable speed control for electric motors. The Drives business is a key factor in sustainability in the industry and society as a whole since the products provided by Danfoss Drives provide energy-efficient solutions in making the world more sustainable.

Also, in Finland in 2023 Danfoss' "Fossil Free Future" program received 10 million euros in funding from Business Finland for fossil-free solutions in industry and transportation. The Fossil Free Future -program is divided into three areas:

1. Highly efficient smart power electronics for electrification.

---

[6] Danfoss. 2023. Accessed 15.05.2023.
[7] United Nations. Accessed 15.05.2023.

2. Electrification for mobile work equipment, smart energy storage solutions and charging infrastructure.
3. Increased efficiency and cost-competitiveness for green hydrogen production. [8]

[8] Danfoss. 2023. Accessed 15.05.2023.

2 RELEVANT TOOLS AND TECHNOLOGIES

This section goes through some basic information about the tools and technologies used in this thesis. More detailed information about some of these topics is given in the next sections.

2.1 FPGA

FPGA (Field Programmable Gate Array) is a programmable device composed of semiconductor components. Notably FPGA offers many key characteristics that make it stand out in the field of electronics design. These include programmability, cost advantages resulting from not needing components to modify the design, high-speed performance due to parallel data processing and better time-to-market due to troubleshooting ease. In this section we will go through some specific parts of FPGA that are relevant for this thesis. In Figure 3 a simplistic representation of FPGA is illustrated. The implementation in this thesis is done for FPGA and investigates the possibilities of FPGA development now and in the future.

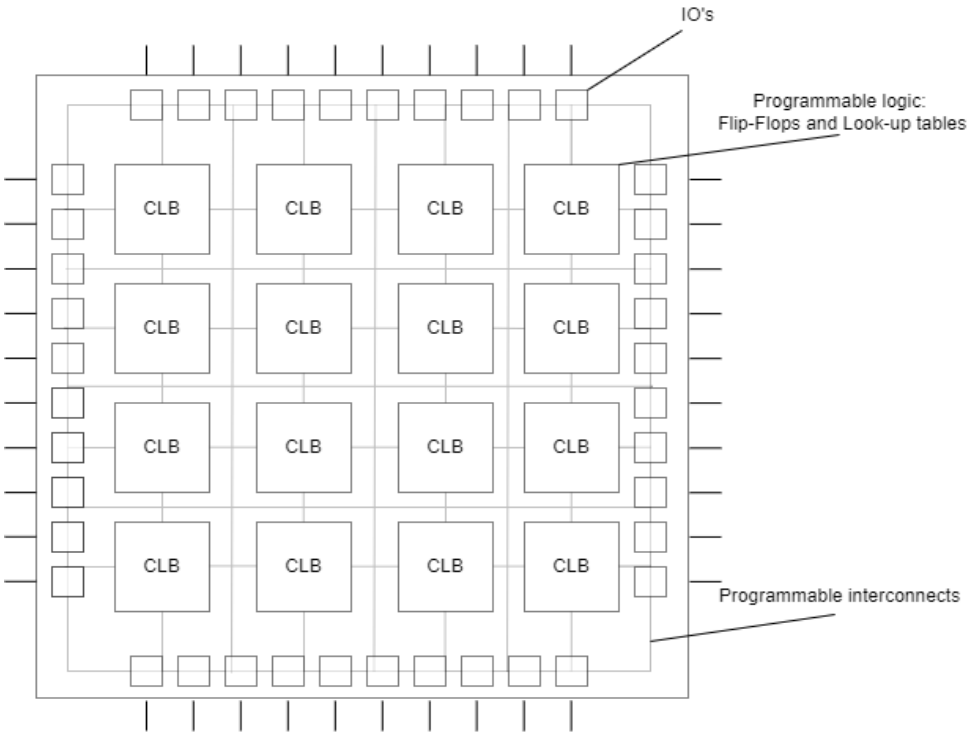

Figure 3. Simple illustration of FPGA.

2.1.1 Key Elements Relevant to This Thesis

IOs are the inputs and outputs of the FPGA device which allows the user to connect the FPGA device to other devices. Typically, an FPGA has some specialized IOs for specific purposes, for example, clock IOs for synchronizing with an external clock. An FPGA has a finite number of IO's and typically the IO's are not to be wasted for unnecessary functions.

LUT (Lookup table) is a key element in FPGA. Essentially, it is a truth table in which different combinations of inputs use different functions to yield output values. For example, an LUT can produce a truth table that produces the output of an AND gate and that table is stored in small RAM (Random access memory). The size of a LUT depends on the number of input bits it can take. For example, a 4-input LUT has 16 possible input combinations. Basically, a lookup table is a customizable truth table that is stored and loaded from small RAM based on the instructions given by the user. [9]

DFFs (D Flip-Flop) are a way to keep a specific state inside the FPGA. D Flip-Flop is illustrated in Figure 4. The labeled pins are D: data input, Q: data output, clk: clock and rst: reset. Data comes from the "D" pin and on the rising edge of the clock signal it is stored to output "Q". So, DFFs are a way for the FPGA to keep a specific state, meaning it is a type of memory. Typically, it stores the input and output on the rising edge of the clock signal until the next rising edge. The Flip-Flop can also store the complement of the stored output value usually marked as output "$\overline{Q}$".

[9] Hardwarebee. Accessed 15.05.2023.

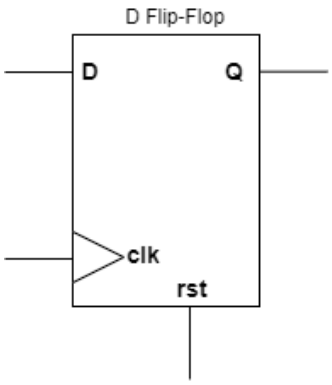

Figure 4 D Flip-Flop illustration.

The next relevant topic is BRAM (Block random access memory) which are a conventional instrument to store large amount of data inside the FPGA. Usually, the price of FPGA correlates with the amount of BRAM in the device.

2.2 RISC-V ISA

2.2.1 ISA

ISA (Instruction set architecture) is an abstract model of a computer and defines how the CPU is to be controlled by the software. It is an interface between the hardware and software and dictates what and how the CPU is doing. An ISA defines for example, the supported data types, instructions, and registers.

2.2.2 RISC-V ISA

RISC-V is an open-source ISA designed by University of California, Berkeley. It was founded in 2015 as the RISC-V Foundation and is incorporated as RISC-V International Association (a non-profit organization that develops and promotes the RISC-

V instruction set architecture) in Switzerland in 2020. RISC-V International includes over 3000 members across 70 countries and the ecosystem is expanding. [10]

The ISA is built avoiding implementation details as much as possible so that it can be used for a wide variety of implementations without it being a specific design for a single or a small group of implementations. The user can also add their own instructions to the instruction set and therefore RISC-V is very sustainable and portable. Next, some goals in defining the RISC-V ISA according to "The RISC-V Instruction Set Manual" by Andrew Waterman, Kriste Asanovic and SiFive Inc. are listed:

- A completely open ISA that is freely available to academia and industry
- A real ISA suitable for direct native hardware implementation, not just simulation or binary translation
- Avoiding "over-architecting"
- 2008 IEEE-754 floating-point standard support
- ISA extension support
- 32-bit and 64-bit address space variants for applications, operating system kernels and hardware implementations
- Highly parallel multicore or manycore implementation support including heterogeneous multiprocessors
- Fully virtualizable
- Simplifying experiments with new privileged architecture designs [11]

RISC-V and its many variants have laid a solid foundation under vendor-independent development of both hardcore and softcore CPUs. Hardcore CPUs are physical processors that are implemented as a dedicated piece of hardware. Softcore CPUs on the other hand are processors that are implemented using programmable logic

[10] RISC-V. Accessed 22.03.2023.
[11] RISC-V. 2017. Accessed 22.03.2023.

(for example, FPGA) and therefore, softcore CPUs are more flexible and can be customized to meet specific requirements.

2.3 SBT

SBT is an interactive build tool for Scala and Java projects. Although the tool can be used for Java projects, it is used more on Scala projects since it has some Scala specific features, for example, the ability to cross build a project against multiple Scala versions. This makes is handy to use with big projects where version control can be harder.

In this project SBT was used to generate the Verilog/VHDL file from SpinalHDL files which are in fact stored as Scala files. SBT allows an easy way to build these SpinalHDL files and generate a ready to use Verilog/VHDL file with just a single command.

2.4 Software Tools (OpenOCD, GDB and Eclipse)

2.4.1 OpenOCD

For the debug session between the implementation and user an OpenOCD session needed to be established. OpenOCD is a tool used for debugging, in-system programming, and boundary-scan testing for embedded devices.

In our case the OpenOCD works through USB-JTAG-Dongle and establishes a debugging connection between the IDE and the SoC (System on Chip, an integrated circuit) inside the chip. OpenOCD needs some configuration specification given in separate configuration files. These specifications include, for example, adapter driver specifications and TAP (Test Access Port, is a part of logic through which the user can access the logic inside the circuit) specifications for JTAG connectivity. [12]

[12] OpenOCD. 2023. Accessed 24.03.2023.

2.4.2 GDB

GDB (GNU Project Debugger) allows us to see what is happening inside OpenOCD and therefore, it helps with software debugging and the connection establishment since we can see what the OpenOCD is logging if it crashes.

In this project GDB is running parallel with OpenOCD to keep log of what is happening inside of OpenOCD and in a sense runs the OpenOCD and works as the top layer of the connection. [13]

2.4.3 Eclipse Embedded CDT

The Eclipse Embedded CDT (C/C++ Development Tools, formerly GNU MCU/ARM Eclipse) was used in this project to maintain and debug the software that was ran on the processor/SoC. This IDE was chosen for this thesis mainly because of its straightforward nature and ease of use when importing makefile projects and building them. Plus, the VexRiscv github has a walkthrough on “Using Eclipse to run and debug the software” [14]

2.5 Multi-vendor Synthesis Tool

An EDA tool for generating the output files for different vendors was used. This tool is used to do the compilation of source design files and synthetization for these files. For place and route and bitstream generation the tool just calls for the vendor’s tool (for example in the case of Xilinx it calls for Vivado) to do the remaining steps for complete implementation. The EDA is there mainly to ease the workflow and to aid in synthetization process making it easy to proceed in the research.

---

[13] Sourceware. 2023. Accessed 24.03.2023.
[14] Github Vexriscv. 2023. Accessed 24.03.2023.

3 THE DESIGN

This chapter reviews the implementation of generic HDL modeling and its steps. The implementation is simple, its purpose is only to act as a test piece for different tools. It has a few necessary logic blocks for functionalities and a processor. The processor is a VexRiscv softcore processor which is generated from a ready-made Scala file written in the SpinalHDL language. At this point the design files for the project are developed before using the translation tool. Figure 5. illustrates the roadmap for project (blue indicating “in progress” status).

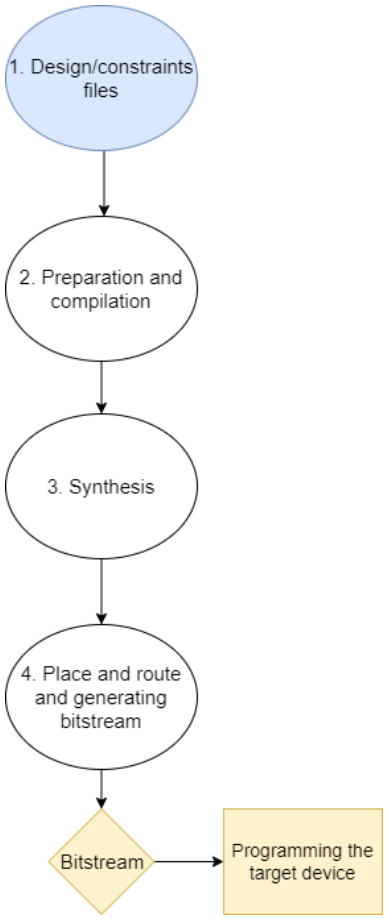

Figure 5. Workflow (blue indicating "in progress" status).

3.1 SpinalHDL and VexRiscv

SpinalHDL is an open-source high abstraction HDL. SpinalHDL is used in this thesis, to generate a VHDL or Verilog file for the processor/SoC.

VexRiscv (rv32im CPU) is an open-source FPGA compatible RISC-V open instruction set based processor implemented with SpinalHDL. [15] VexRiscv was selected for the implementation of this thesis due to its openness and scalability. VexRiscv also won the RISC-V SoftCPU contest in 2018 RISC-V summit where it was the highest-performance Microsemi implementation and the third smallest Lattice implementation [16]. In this implementation, the VexRiscv processor plays a key role, especially due to its vendor-independent features and support for FPGA development. VexRiscv is licensed under the MIT-license [17]. The MIT-license grants a permission to any person obtaining a copy of the software and associated documentation files, to deal in the software without restriction. Meaning a user with a copy of the software and its documentation can do anything they want to the software and its documentation if the license notice, and the original copyright are included. [18]

The VexRiscv repository comes with two independently working SoCs a so called Briey SoC and Murax SoC. The Briey SoC is more complicated and has a lot of different features and it is over twice the size of the Murax SoC. Because of these reasons the Murax SoC was deemed a better fit for this use case. Since it needed minimal effort and modifications to work as a test piece for this thesis.

---

[15] Github Vexriscv. 2023. Accessed 19.04.2023.
[16] RISC-V. 2018. Accessed 19.04.2023.
[17] Github Vexriscv. 2023. Accessed 19.04.2023.
[18] United Nations. Accessed 19.04.2023.

VexRiscv repository also has great documentation and many features have already been implemented for different vendors or completely vendor independently and therefore it is easy to find solutions in case of any issues in implementation.

3.2 Murax SoC

As mentioned earlier, the implementation is strongly based on the SoC built for the processor (VexRiscv) called Murax. This SoC is a simple entity, so it fits well as the base of this implementation. In addition, it has all the necessary parts and functionalities for this thesis.

The size of the SoC on different vendors devices can be seen in Figure 6. According to the VexRiscv repository documentation [15]. It is important to note that the sizes are not mutually comparable since different vendors devices might have some architectural differences and therefore the sizes must only be taken as directional information. The sizes should only be compared to same vendors devices when no further information is given.

```
Murax interlocked stages (0.45 DMIPS/Mhz, 8 bits GPIO) ->
    Artix 7     -> 216 Mhz 1109 LUT 1201 FF
    Cyclone V   -> 182 Mhz 725 ALMs
    Cyclone IV  -> 147 Mhz 1,551 LUT 1,223 FF
    iCE40       ->  64 Mhz 2422 LC (nextpnr)
```

Figure 6 The size of the Murax SoC on different vendors devices.

Murax SoC has the following specifications and results:

- 0.37 DMIPS/MHz (Dhrystone MIPS (Million instructions per second))
- 8 kB of on-chip ram
- JTAG debugger (eclipse/GDB/openocd ready)

- Interrupt support
- APB (Advanced Peripheral Bus) bus for peripherals
- 32 GPIO pin
- one 16 bits prescaler, two 16 bits timers
- one UART (Universal Asynchronous Receiver Transmitter) with tx/rx fifo [19]

3.3 Modified Murax SoC

Some changes were made so that the SoC would suit this thesis better. One unused general-purpose output has been removed and some changes to the JTAG block was done to ease the debugging. The changes were made on the existing SpinalHDL file (Murax.Scala). This file is a SpinalHDL representation of the SoC and it's used to generate a Verilog/VHDL file depending on user preferences. In this case a Verilog file "Murax.v" is generated with this Scala file using SBT. Figure 7. shows a block diagram, from which the most important blocks for this thesis and their connections can be observed, such as RAM memory, processor, JTAG adapter and GP (General Purpose) output, UART and a timer block with interrupt functionality connected via APB bridge and APB decoder. This implementation blinks an LED with periodic interrupt. The timer counter logic generates an interrupt by user specified intervals to the CPU, and this causes the LED to change state (on/off). This is done to provide visual feedback of the system activity in real-time.

[19] Github Vexriscv. 2022. Accessed 19.04.2023.

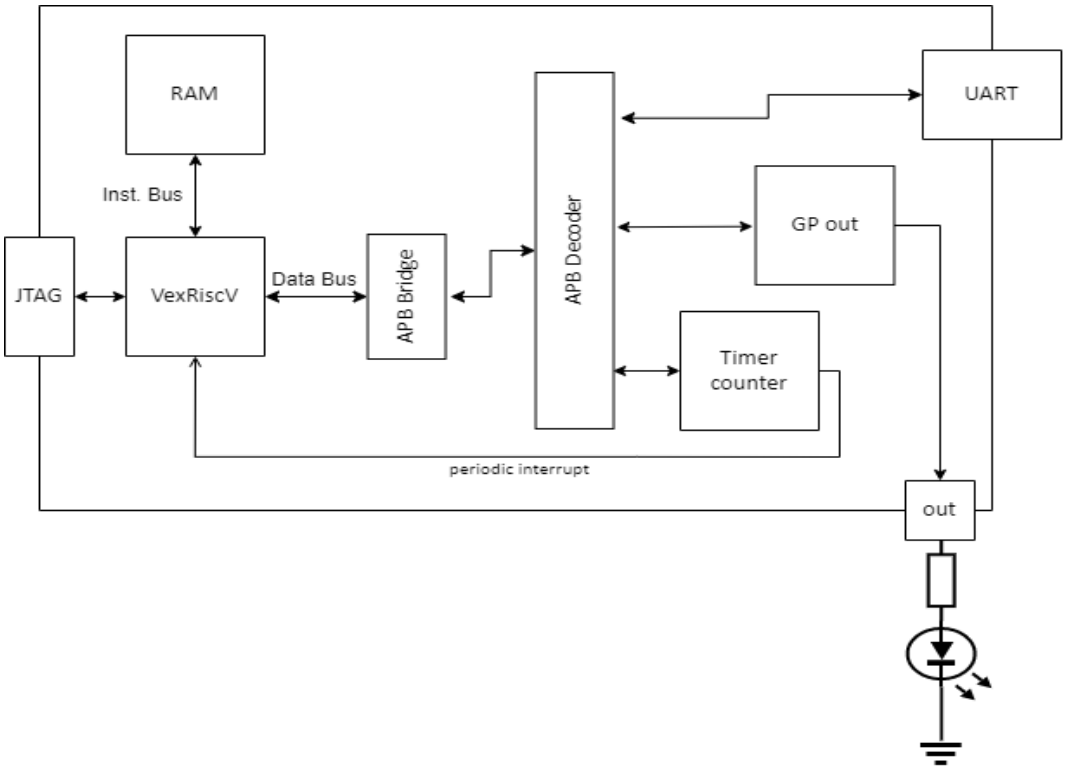

Figure 7. A block diagram of the implementation.

3.4 Blocks and Their Functions

The logic blocks used in this implementation and their operating principles have all been implemented using generic HDL with as vendor-independent methods as possible. In the next four sections some of the more important blocks and their functions are described.

3.4.1 JTAG

JTAG block with its own JTAG bridge was implemented so that software could be debugged and ran on the device. Murax SoC by default implements its own JTAG TAP on a specific TAP. A problem occurred when trying to debug and run the software on the device using only one USB to JTAG cable. Therefore, some modifications were needed on the SoC so that it implements a bridge between the SoCs

JTAG tap and the native JTAG. This is done only to ease the implementation since without this change a secondary JTAG cable for the SoC inside the FPGA would be needed. This JTAG debugger block is otherwise fully suitable for this thesis since it is OpenOCD, GDB and Eclipse ready. The changes in the JTAG block are better described in Chapter 3.5.

3.4.2 VexRiscv RV32I[M]

An FPGA compatible 32-bit processor with fully vendor independent design and functionalities is suitable for this thesis. This is the main part of the system and functions as a sole processor for the customized Murax SoC. The processor is done using the open RISC-V ISA and therefore is fully scalable and modifiable. The RV32 means that it is a RISC-V 32-bit variant, and "I" indicates the base integer instruction set and finally the "[M]" indicates the standard extension for integer multiplication and division. Virtually any RISC-V ISA processor with FPGA capabilities would fit this thesis but VexRiscv was chosen due to its documentation and support for multiple vendors.

3.4.3 APB Bus

This SoC has an APB bus for peripheral blocks. This is great for our purpose since it is an open-standard interconnect and is therefore vendor independent. It falls under the so called AMBA (Advanced microcontroller bus architecture) specification more specifically the first version of AMBA specification. It is a low-cost interface with minimal power consumption and reduced interface complexity. [20]

The reduced interface complexity is important since this implementation is made to work as a test piece and therefore needs to not be overly complicated and

[20] ARM developer. 2023. Accessed 19.04.2023.

fancy. This bus interface works well with the implemented system plus it is easy to manipulate if needed.

3.5 Running and Debugging the Code on the SoC

The software was built in Eclipse Embedded CDT as a Makefile project. The software is a ready-made C code that blinks the LEDs. It can be found in the VexRiscv repository [21], and it is made specifically for the Murax SoC and works without changes even though modifications for the SoC were made.

For the debug connection from the user to the SoC, modifications in the JTAG bridge were needed, as mentioned in Chapter 3.4.1. These modifications are made to enable the possibility for debugging using a single USB to JTAG cable. Xilinx has its own solution to this problem, a debug bridge IP called BSCANE2 (A bridge between the native JTAG TAP on the device and the Murax SoC TAP. See Figure 8.), also Intel and Lattice have their own similar yet different answers for this problem. Intel has the Virtual JTAG (VJTAG) and Lattice ECP5 family has the JTAGG primitive cell implementation. These can be used in the translating process to ease the debugging stage.

Figure 8 illustrates the system that is used for the debugging in its entirety. The basic principle is that the output of the SoCs JTAG tap is forwarded through a bridge to the native JTAG on the chip and this allows the communication to go through a single line of connectivity.

[21] Github VexRiscv. 2023. Accessed 19.04.2023.

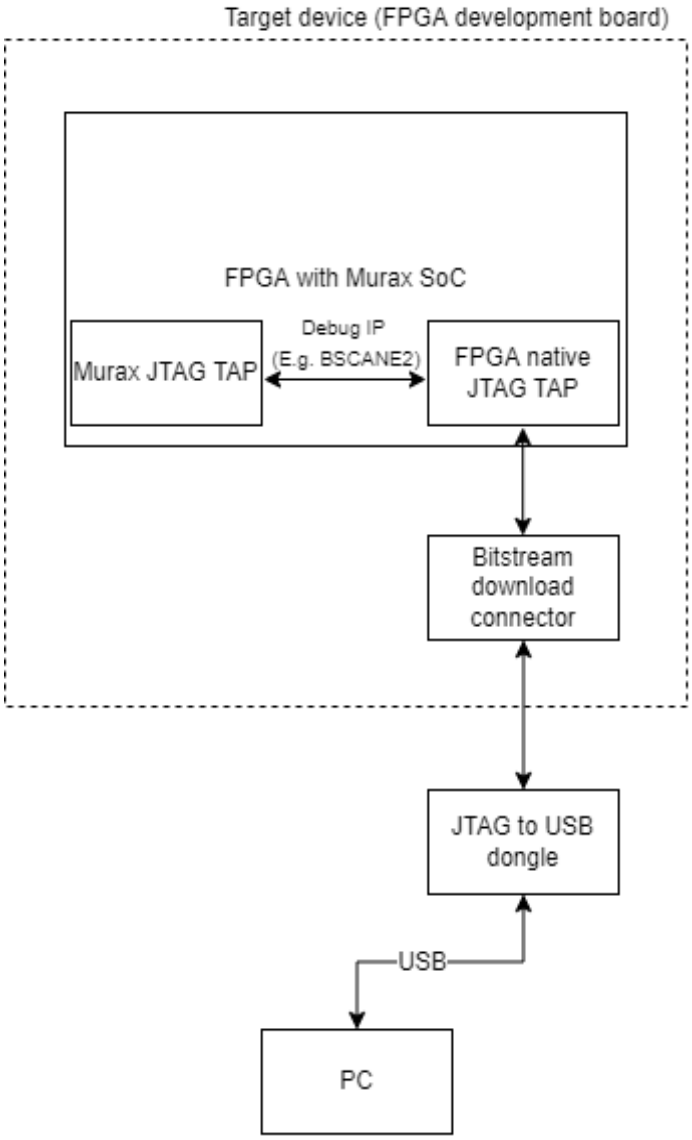

Figure 8. Debugging system connections.

After these modifications OpenOCD connection can be established with needed configuration files for the session. These typically include configuration file for the JTAG connection and a configuration file for the part that is used. Some changes to these configuration files are needed depending on the user's hardware.

After a successful OpenOCD connection via GDB is established, the software can be started and on success the LEDs on the board start to blink.

4 TRANSLATING FOR DIFFERENT VENDORS

This section will go through the steps taken for generating the needed output files for the design for different vendors devices. The implementation was first done for Xilinx (ARTIX-7 7A35TIFTG256-1) and in this section this implementation is translated to Intel (Cyclone V 5CGXFC5C6F27C7). Some abstraction is needed for some vendor specific blocks and some creative ways for implementation are needed. For the generation of the files an EDA tool was used but all the source files stayed the same in functionality (some minor adjustments were made).

4.1 Preparations

Some parts of the project source files are always vendor specific, for example the constraints files. These vendor specific parts must be changed from vendor to vendor. The main issue with vendor independent design in FPGA development is the ability to generate bitstreams for different vendors since they are unique from vendor to vendor. Fortunately, the used EDA tool will be able to do that via generating the scripts for different vendors tools according to the selected device.

First, the customized Murax SoC had a Xilinx specific debug block on it. This was removed for the Intel implementation and no specific debug IP was implemented for the Intel device since the circuit has some general-purpose input/output pins on board so there is no need to only use one JTAG cable. So, the implementation used for the Intel device is even more like the original Murax SoC with its own JTAG TAP. Also, the top level of the project files needed to be changed to use the Murax JTAG TAP without rerouting it.

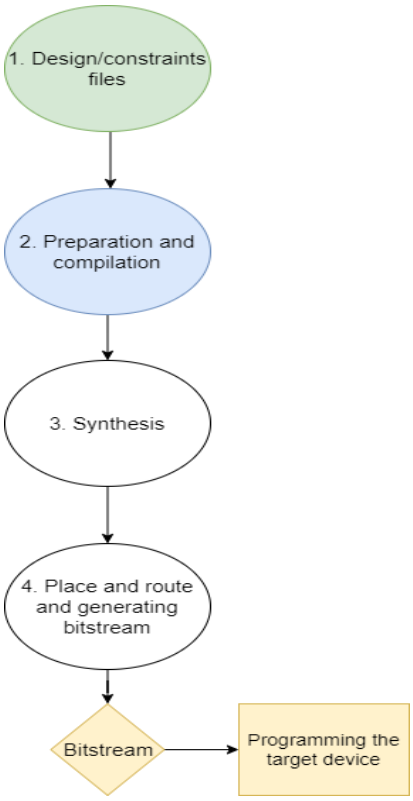

Figure 9. Workflow (Green indicating "done" status).

4.1 Constraints

As mentioned before, the first instance of the design was done on Xilinx, so it has Xilinx constraints (.xdc) files, these needed to be converted to the target device and made sure the pins were compatible on the target device. The target device is an Intel Cyclone V family device and therefore the constraints must be suitable for Intel. This was done using the EDA tools constraints editor which essentially generates a suitable file according to user preferences. In the case of Intel an SDC file is generated containing the constraints specified by the user.

The constraints for this implementation are simple since the top level wraps the inputs and outputs neatly and there are not many clocks to keep track of. The pins specified in the constraints are shown below.

- Inputs
  - user_sw0 – user_sw3, software pins
  - user_btn0 – user_btn3, button pins
  - JTAG input pins (tck, tms, tdi, and trst)
  - serial_rx, UART input
  - cpu_reset (active low)
- Outputs
  - user_led0 – user_led3, for the LEDs
  - serial_tx, UART output
  - JTAG output (tdo)
- Clocks
  - clk_main (50MHz)

4.2 Compilation

The source files (including the Murax, toplevel and possible constraints) are imported into the EDA tool and the files are compiled so that the RTL view of the EDA tool can be used to possibly modify connections etc. This functionality was not used in this thesis and the tools used in the development were mainly used as text editors for the source files and no graphical user interface was used to develop the project or SoC.

After the compilation process the constraint editor in the tool can be used to edit the timing constraints if needed. In the case of this thesis the EDA tools constraints editor was explored and used to generate constraints for the device. This was an easy way to make sure that different implementations had their own constraints in place and there was no mix up between different implementations timing constraints. The user can also import their own constraints file if one is already made which was the case for the Xilinx implementation that was done before using the EDA tool.

4.3 Synthesis

The synthesis along with the compilation is done in the EDA tool without the need of a vendor specific tool. The tools synthesis is basic FPGA synthesis with implementation of operators, optimization of blocks and timing optimizations if needed. The synthesis also produces a lot of reports regarding area usage and possible timing violations.

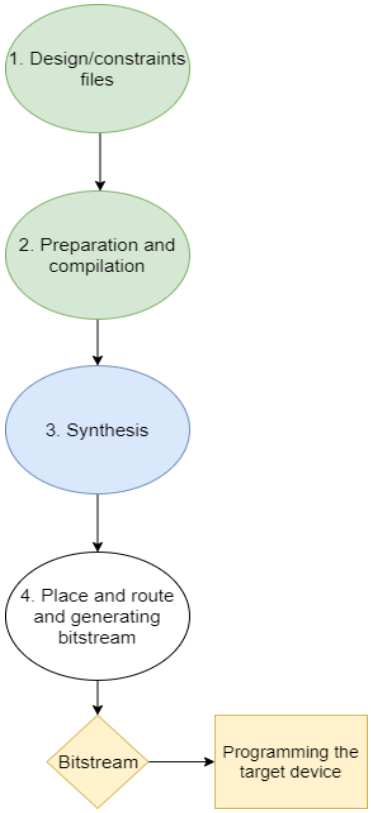

Figure 10 Project workflow.

The synthesized implementation meets the timing requirements of this simple system and there are no timing violations. The area measurements seem also feasible. These results will be investigated further in Chapter 5 (Results of translating process).

This step also generates the needed VQM netlist file for Quartus project if the vendor tool is needed to be used. In this thesis the vendor specific tools were not used

except as a text editor for the initial design but after that every step of typical development was done using the EDA tool. The corresponding netlist file for Xilinx (EDF (Electronic design format) file) is also generated in the case of the Xilinx implementation. This netlist file is used in the next step (place and route) by the vendor specific tool to generate the needed output files for chip programming.

According to the tools timing reports the design produces positive slack time for critical paths and therefore, it does not need any further performance optimization steps and place-and-route can be started.

4.4 Place and Route

The place and route step of the EDA tool runs TCL (Tool command language see Chapter 6) scripts on the vendor tool to generate output files for the corresponding vendor. In this case the tool runs Intel's Quartus to generate all the needed output files and reports. These include the mapper report, fitter report, timing analyzer report and the assembler report. This step also generates the needed bitstream file(s) for the implementation to run on hardware. This generated sof (SRAM object file) file can then be used to program the chip through USB to JTAG cable.

The resulting files and reports of this stage are specified further in the next chapter (Chapter 5). The place and route steps went as expected and results are correct. The implementation does not produce negative slack and the timings are functional. Also, the pin mapping and fitter report look feasible, so the place and route seem to have been successful. The file for programming the device is now generated with this tool flow for both vendors and the devices can be programmed. The translation process has reached its goal.

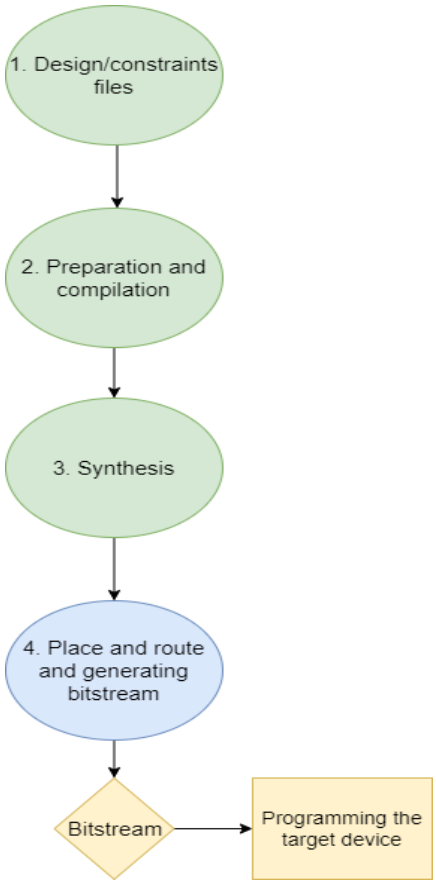

Figure 11 Project workflow.

Since the Intel implementation was done without a debug bridge IP the debugging and running software would be done with two JTAG to USB cables. One for the bitstream and one for the software as illustrated in Figure 12.

The place and route step was also done for the Xilinx counterpart without the BSCANE2 changes. As all the other steps for comparison purposes so that both implementations were done with all the same software and tools. The normal JTAG block for comparison and the results will be presented in the next chapter (Chapter 5).

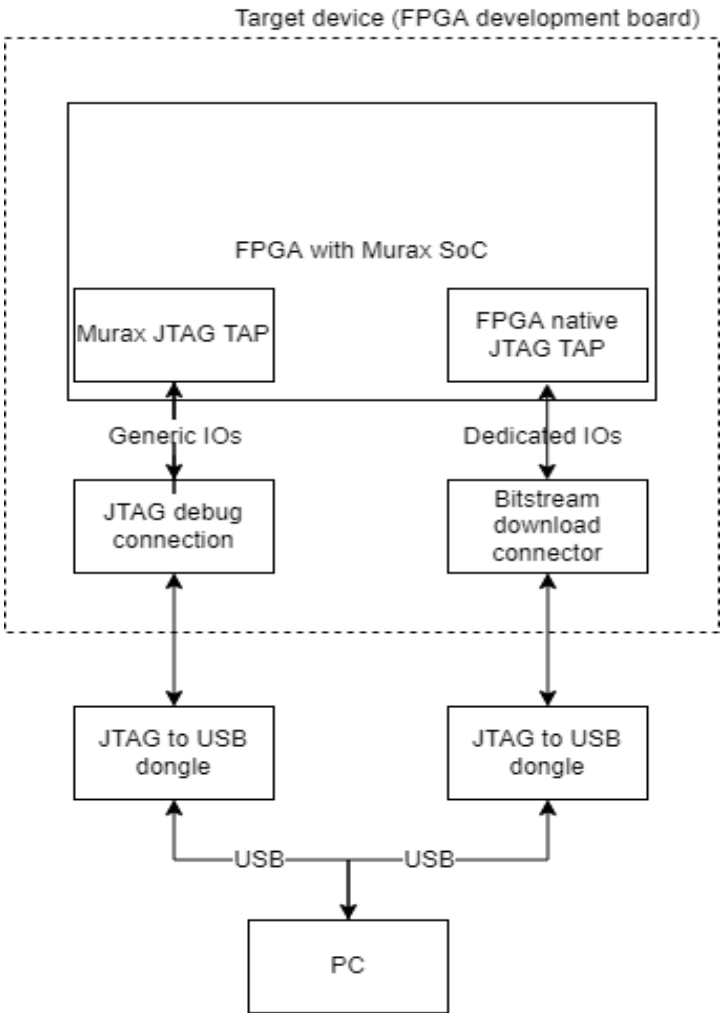

Figure 12. Running and debugging software without the debug bridge IP.

5 RESULTS OF TRANSLATING PROCESS

This section will go through the results of the translating process in more detail. Also, this section will recap what is the results of this project work and if there is something left desired for the process.

5.1 Resulting Reports

The implementation is translated to two different vendors devices: Xilinx ARTIX-7 7A35TIFTG256-1 and Altera Cyclone V 5CGXFC5C6F27C7. The translation was successful, and the implementation is working as expected. As can be seen from the last chapter, the workflow for translating this vendor independent design was straight forward and simple. Since the design is vendor independent no further adjustments for the translation was needed for it to work properly.

The place and route step generated some reports of the implementation (mapper report, fitter report, timing analyzer report and the assembler report). In this section those will be investigated a little further to see any possible differences between vendors.

The generated reports (shown in the EDA tool) differ between vendors since different tools produce different types of reports by default. This might lessen the need or will to use a vendor independent EDA tool if the user would still want to open vendor specific tool to get their hands on a specific report. These sorts of challenges will be investigated in Chapter 6. (Alternative tool flows for translation).

5.1.1 Area Reports

One of the reports the EDA tool generates is so called area usage report. This report shows high level information of the area taken from the chip by the implemented logic. The results of the area reports are as expected, and the implementation is taking a small area of the chip since it is a simple design.

According to the area measurements provided by the VexRiscv repositorys documentation, the results seem similar. Some deviation can be explained by the small modifications done to the SoC. Therefore, the SoC was also implemented without any changes so that the results can be compared more easily.

According to the tools area usage reports the following is used (Cyclone V 5CGXFC5C6F27C7):

Table 1. Cyclone V 5CGXFC5C6F27C7 area report.

	IOs	ALMs(estimated)	LUTs	Registers	Memory bits	DSP Blocks
Used	21	931	1243	1277	35072	0
Available	364	29080	-	-	4567040	150

Table 2 is the area report results for the Xilinx counterpart (ARTIX-7 7A35TIFTG256-1) which was the first implementation of this thesis. The results of table 2. are for the implementation with the Xilinx debug IP (BSCANE2) and therefore there is some deviation. In Table 3 the results of the exact same implementation with Xilinx chip as in Table 1 can be found.

Table 2. ARTIX-7 7A35TIFTG256-1 with BSCANE2 area report.

	IOs	Global Buffer	LUTs	CLB Slices	DFFs or Latches	Block RAMs
Used	21	1	1017	156	1241	2
Available	210	32	20800	8150	41600	50

Table 3. ARTIX-7 7A35TIFTG256-1 without BSCANE2 area report.

	IOs	Global Buffer	LUTs	CLB Slices	DFFs or Latches	Block RAMs
Used	21	2	1029	161	1282	2
Available	210	32	20800	8150	41600	50

The results are as expected and there is no unexplainable deviation to the documentation given in VexRiscv repository. The Cyclone V implementation uses more LUTs than the Artix-7 version. This is most likely due to architectural differences between these two vendors (Intel and Xilinx), since the synthesis tool is the same and design constraints are the same (there is no tighter timing constraints etc. on one of the implementations) for both vendors devices. Also, the LUT structure is the same across the devices (6-input LUTs).

5.1.2 Utilization Reports

More detailed utilization reports are generated by the vendor specific tools that are ran with the TCL scripts the EDA tool generates. These scripts can be modified if extra configurations are needed/wanted. These utilization reports are investigated in this section through the lens of vendor independency in FPGA development, meaning the details are not as important as the relation between vendors implementations.

In utilization reports the placements and area usage are presented in more detail for example, used primitives and I/O bank information are specified in these documents. First, the fitter reports of the Cyclone V implementation are investigated and then that is compared to the initial Xilinx implementation.

The Cyclone V implementations fitter reports are as expected, and they correspond with the area reports provided earlier with a little more detail. Timings and routes are feasible and functional. The whole process of translating the initial design to Cyclone V 5CGXFC5C6F27C7 seem to have been successful and this can be demonstrated with programming the board in question with the SOF file provided by the place and route step. Figure 13 shows the activated LEDs (off in the picture). This demonstrates that the file is programmed successfully, and all the corresponding pins are activated as was configured.

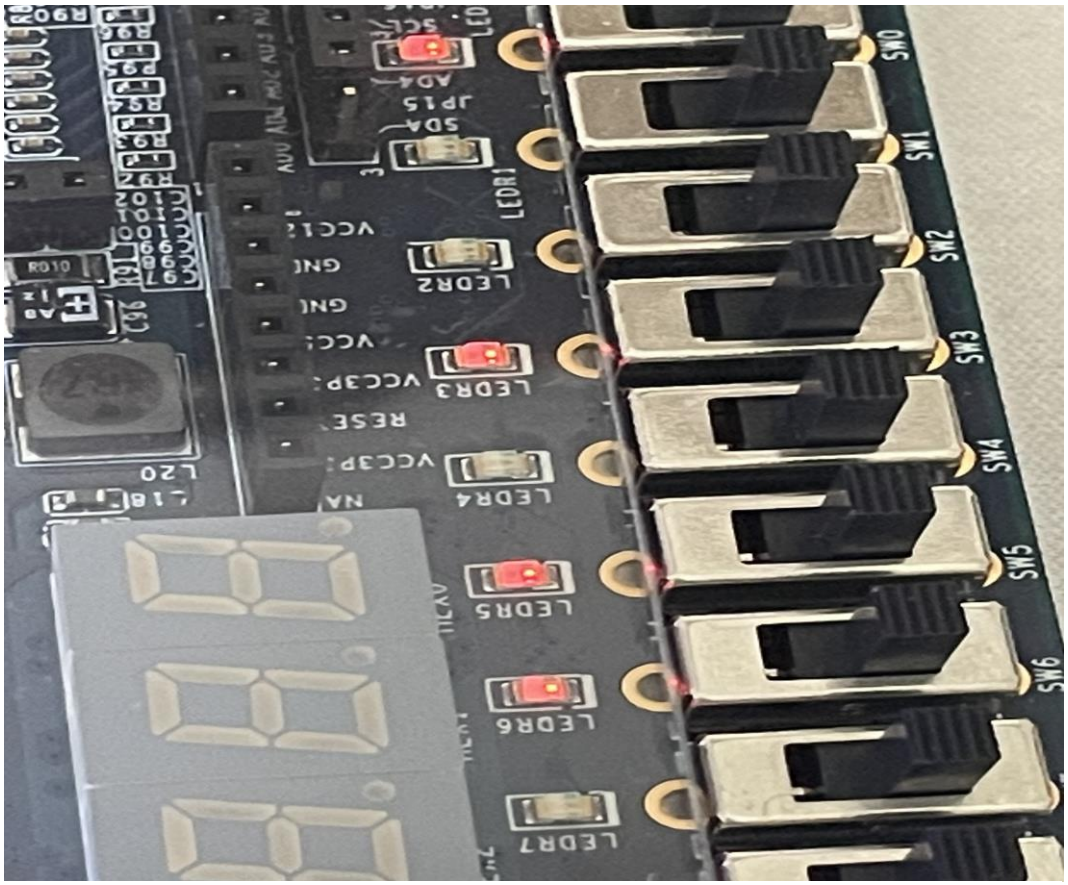

Figure 13. A still image of the activated LEDs on the board (Cyclone V 5CGXFC5C6F27C7).

5.1 Reflections on the Process

The translation of the implementation from the source files (Murax.v and top-level.v (wrapper)) went as expected and results are correct. The process on that regard can be considered successful. The workflow with the EDA tool was simple to use and easy to execute as it resembles most the workflow of other FPGA development tools, for example, Vivado or Quartus.

The process of translating this simple implementation from a vendor to another was easy to execute. The only major difference the implementations of these two vendors was the constraints since the main source files were fully vendor independent. In a bigger project with different vendor specific IPs the vendor specific parts would need to be built by the developers so that the parts are no longer vendor specific this would accumulate some extra work. For example, if some vendor specific IP-cores are used in the initial project from a vendor's library those IP-cores would need to be found also from the other vendor's library and this produces possibly very different designs and software implementations from vendor to vendor. Therefore, a better solution would be a generic HDL representation of the IP-core in question and use that on all the projects. This obviously produces some development work.

Additionally, for different clock speeds some PLL defining logic would need to be implemented. For example, a conditional statement that implements the logic for the vendor in question. This statement/generating cycle would need to be implemented in the project build phase either with the language used in this thesis (SpinalHDL) or with some other method. An additional method would be that the project HDL would be its own entity and the PLL counterpart would be instantiated after the initial HDL representation with some high-level selection logic or conditional statement.

6 ALTERNATIVE TOOL FLOWS FOR TRANSLATION

In this chapter some alternative tool flows for vendor independent development are investigated and compared to the flow used in this thesis.

6.1 The Need for Alternatives

One reason for exploring alternative methods for translating FPGA designs to different vendors is that using a ready-made EDA tool may have certain limitations. For example, while the tool can perform synthesis and generate a netlist a thorough simulation of the circuit would still need to be done through a vendor specific simulation tool for better functionalities and accessibility. Additionally, even when using an EDA tool, the place and route step still needs to be ran on the vendor specific tool albeit “silently”. These sorts of limitations prompt a question about an alternative method of translation.

6.2 Scripting

The main function of the EDA tool is generating script files for place and route on the vendor’s tool and running those scripts “silently” (if the user so wants) on the tool. This can be done by yourself and already is a part of FPGA development to some extent. Many of the vendor specific tools can be scripted using TCL. TCL is a common scripting language for software tools, and it is easy to use and the support and ecosystem around is vast. But the script generation can be also done using other common languages for scripting for example, Python.

This is how the EDA-tool used in this thesis works and this can be done by the user themself too. Then the project would be fully vendor independent since it would not depend on the support and ecosystem of the EDA tools provider.

6.3 Pros and Cons of Scripting

A major positive side to developing and running the scripts independently is vendor independence. The project would be fully vendor independent and not rely on the EDA tools provider. Different EDA tools have different supports for vendors for example, the newest devices of a vendor may not be included in the tool's catalogue or there might be some delay to when those devices will be added to the catalogue. Many smaller vendors are most likely never included since the demand for support is not great enough for the provider of the tool. By implementing their own scripting, the user can be assured that they can use any vendor they see suits best for the project in question.

Another positive side would be the cost. Using a licensed EDA tool costs quite a lot annually when considering that usually a company needs more than one license per development team so that many developers (preferably all) can use the tool simultaneously.

One major disadvantage of self-scripting the vendor specific tool is obviously that it is time consuming. This type of method would most definitely be very time consuming regarding the alternative method of using a tool that generates the scripts for vendor specific tools "at a push of a button". In a company setting time is money and since this way would drive up the time needed for development it would need more money put into it. A way to decrease the time spent on implementing scripts for tools is to use an open-source library of scripts. These are readily available for most vendor specific development tools. For example, a FPGA specific Python based script tool called PyFPGA [22] or for more comprehensive project management: Edalize [23].

[22] Github PyFPGA. 2023. Accessed 03.05.2023.
[23] Github Edalize. 2023. Accessed 03.05.2023.

Also, scripting the vendor specific tools yourself can cause inconsistent methodology across different designs or designers, which can make it difficult to verify and maintain the design. This could be avoided with good documentation and specification regarding the scripting meaning this type of workflow would accumulate extra work for developers. Compared to using a single tool for the translation where the specifications would always remain consistent.

6.4 Comparing to a Ready-made EDA Tool

Compared to a ready-made EDA tool the scripting method of translating is more time consuming but cheaper upfront since there would be no need for additional license fees (vendor specific tools would need to be purchased) and more vendor independent since the user is not dependent on the provider of the tool.

On the other hand, a vendor provided EDA tool would be ready to use after purchase but may have limitations in vendor support for some FPGA vendors. This would most likely be negotiable if the company buying the licenses is a valuable customer for the provider of the tool. Also, the price of licenses is negotiable but companies that deal with software and products in the information technology field usually have many different licenses to upkeep and these cumulative costs are high, therefore it is a formattable step to take in even more licensed products. Although, the usage of a single EDA tool for all other steps except the place and route would possibly allow the user to use license free versions of the vendor specific tools which would possibly even out the price.

6.5 Personal thoughts

My personal thought on matter is that the scripting of the tools by the developers themselves would be preferable since then there would be no need for extra tools in the workflow and typically vendor specific tools are already scripted at least to some extent to ease the workflow. Also, since developer teams usually already

have scripts for the tools the only remaining challenge would be project management between vendors. This can be handled with exact file structures and specified way of working, so that the project files are organized and named well.

Vendor independency could be achieved with writing as generic HDL as possible plus having good command of the project structures and different vendors projects. The latter could be done with a simple file structure which differentiates the implementations between vendors but have them under the same project header. For example, with a project the project folder would have different implementations under it in their own folders with corresponding vendor describing names (illustrated in Figure 14). Assuming the initial HDL representation is fully generic and vendor independent, it could be synthesized and implemented with ease for different vendors by running developed scripts for different vendor specific tools. Additionally, if the implementation has some vendor dependent parts those versions should have their own HDL representations for different vendors.

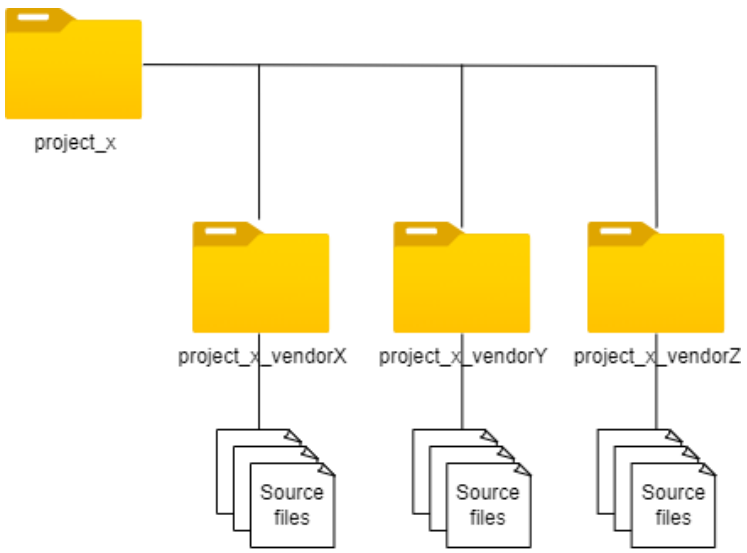

Figure 14. Illustration of the possible filestructure of a vendor independent project.

Another way for implementing vendor independent parts would be a MakeFile type implementation with a build file for specifying different vendor specific parts and all other additional configurations if needed. Makefile is a Make-tools text format configuration file. Make is an automation tool which can be used to dictate, for example, which parts of the project need to be recompiled. Make can be used to automate bigger project flows easily [24]. This would ease the workflow and upkeep of a project for different vendors since all the configurations would be done on the same frame and therefore it would be more straight forward to maintain many different vendor implementations of the same HDL representation.

All things considered the scripting of the tools by the user themselves is obviously a bit more arduous but after the initial steps are taken and the environment is working it would be a better way to tackle a multivendor supporting project in FPGA design in a company setting. Be it that it is doable with an EDA tool with some limitations.

[24] Linux.die.net. make(1) - Linux man page. Accessed 15.05.2023.

7 CONCLUSIONS

This section will try to find answers to the questions asked in the beginning of this thesis. What challenges are encountered and how can they be avoided? Is it feasible to strive for vendor independence? Also, this section will go through what this means for the future and some future research ideas.

The thesis project to implement the same design on couple of different vendors and mapping the possible flow of vendor independent FPGA development using an EDA tool went well and the questions mentioned above were answered by the process. Also, an additional method of working vendor independently was investigated and deemed feasible to use as well.

7.1 What Challenges are Encountered and How They Can Be Avoided

Some challenges were expected since working fully vendor independently is quite an ambitious goal. From the design side the obvious challenge was how to work fully vendor independently regarding designing with generic HDL only. This way of working requires some sort of vendor independent CPU to be implemented to reach the goal of vendor independency. Fortunately, there are already many possible candidates that use an open ISA and in the future this ecosystem will undoubtedly grow. One key problem with vendor independent development is some always vendor dependent parts such as I/Os, constraints and PLL.

I/Os and constraints can be automated quite easily with scripts but PLL being a physical component implementation is a bit trickier to approach, also some vendor specific IP-cores can be a challenge. Some possible solutions are addressed in Chapter 5.1. Lastly, one challenge is the EDA tool itself. Using this kind of tool lessens the vendor independency of the project since the tool is most likely provided by a vendor and therefore, the project is dependent on this provider. This challenge is addressed in Chapter 6.

7.2 Feasibility of Vendor independence

The methods investigated in this thesis will have an impact on the ease of development and upkeep of FPGA projects. Some new types of development would need to be implemented regardless the way the users decide to go (ready-made EDA tool or scripting the tools themselves). Vendor independence would still in my opinion be desirable since it eliminates or at least minimizes the risks involved in development where the user is dependent on chip vendor(s). Also, it gives great negotiation leverage when dealing with chip vendors since the user can always just choose the cheapest alternative with the specifications suitable for the project.

In the light of this thesis, it is feasible to strive for vendor independence using these methods, at least to some extent. It is also worthwhile at least to investigate the possibilities within the group/company in question if it suits their needs at the moment. The ecosystem is there, even though it is not as vast as for proprietary devices and projects, its growing via open-source projects and collaborations between companies and organizations striving for vendor independence.

7.3 General Thoughts

The goal of the thesis was to investigate the possibilities of vendor independent FPGA development. This investigation was done by implementing a simple design for a couple of vendors devices via an EDA tool. This investigation was successful, and another way of working vendor independently emerged from that investigation. This investigation can be considered a success since it answered the questions asked in the beginning of the process.

This thesis was a step into an unknown land for me and others involved in the sense that vendor independent methods in FPGA development are still somewhat new and for example RISC-V has gained a lot of its notoriety recently. Because of these reasons the thesis work was very interesting and exciting. I also got to use a

lot of the things I have learned in university and especially as a trainee working with FPGAs. Yet I learned many things during this project, and it has been very fruitful regarding my professional future.

7.1 Suggestions for Future Research

For future research ideas at least, the method described in Chapter 6. (scripting the vendor specific tools) could be investigated further and some bigger projects with more vendor dependent premises would be interesting to research via these methods. Also, even more comprehensive research into the RISC-V ISA would be beneficial for the future of embedded development not only in the scope of FPGA development but as software development in the embedded field. This would give a better insight to what is possible in the field of open instruction set architectures regarding performance and scalability.

REFERENCES

AMD. 2022. AMD Completes Acquisition of Xilinx. Press release. https://www.amd.com/en/press-releases/2022-02-14-amd-completes-acquisition-xilinx Accessed 22.03.2023.

ARM developer. 2023. AMBA APB Document Specification. https://developer.arm.com/documentation/ihi0024/c/Introduction/About-the-APB-protocol Accessed 19.04.2023.

Arstechnica. 2023. Google wants RISC-V to be a "tier-1" Android architecture. https://arstechnica.com/gadgets/2023/01/google-announces-official-android-support-for-risc-v/ Accessed 22.03.2023.

Danfoss. 2023. Danfoss to receive EUR 10 million in funding from Business Finland for fossil-free solutions in industry and transportation. https://www.danfoss.com/en/about-danfoss/news/cf/danfoss-to-receive-eur-10-million-in-funding-from-business-finland-for-fossil-free-solutions-in-industry-and-transportation/ Accessed 15.05.2023.

Danfoss. 2023. Danfoss at a glance. https://www.danfoss.com/en/about-danfoss/company/danfoss-at-a-glance/ Accessed 15.05.2023.

European commission. MIT-license. https://joinup.ec.europa.eu/licence/mit-license Accessed 19.04.2023.

Github. 2023. Edalize documentation. https://github.com/olofk/edalize Accessed 03.05.2023.

Github. 2023. Hello_world Murax SoC C-code. https://github.com/SpinalHDL/VexRiscv/tree/master/src/main/c/murax/hello_world/src Accessed 19.04.2023.

Github. 2022. Murax.scala file. https://github.com/SpinalHDL/VexRiscv/blob/master/src/main/scala/vexriscv/demo/Murax.scala Accessed 19.04.2023.

Github. 2023. PyFPGA documentation. https://pyfpga.github.io/pyfpga/ Accessed 03.05.2023.

Github. 2023. README.md. https://github.com/SpinalHDL/VexRiscv Accessed 19.04.2023.

Github. 2023. Using Eclipse to run and debug the software. https://github.com/SpinalHDL/VexRiscv#using-eclipse-to-run-and-debug-the-software Accessed 24.03.2023.

Github. 2023. VexRiscv license. https://github.com/SpinalHDL/VexRiscv/blob/master/LICENSE Accessed 19.04.2023.

Hardwarebee. Overview of Lookup Tables (LUT) in FPGA Design. https://hardwarebee.com/overview-of-lookup-tables-in-fpga-design/ Accessed 15.05.2023.

Intc. 2022. Intel Launches $1 Billion Fund to Build a Foundry Innovation Ecosystem. https://www.intc.com/news-events/press-releases/detail/1525/intel-launches-1-billion-fund-to-build-a-foundry Accessed 22.03.2023.

Lattice. 2019. Lattice and SiFive Announce Collaboration to Allow Lattice FPGA Developers Easy Access to RISC-V Processors. https://www.latticesemi.com/en/About/Newsroom/PressReleases/2019/201938LatticeandSiFive Accessed 22.03.2023.

Linux.die.net. make(1) - Linux man page. https://linux.die.net/man/1/make Accessed 15.05.2023.

OpenOCD. 2023. OpenOCD User's Guide. https://openocd.org/doc/pdf/openocd.pdf Accessed 24.03.2023.

RISC-V. About page. https://riscv.org/about/ Accessed 22.03.2023.

RISC-V. 2017. The RISC-V Instruction Set Manual Volume 1: User-Level ISA. https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf Accessed 22.03.2023.

RISC-V. 2021. Wait, What? MIPS Becomes RISC-V Classic CPU Company Exits Bankruptcy, Throws in the Towel |Jim Turley, Electronics Engineering Journal. https://riscv.org/news/2021/03/wait-what-mips-becomes-risc-v-classic-cpu-company-exits-bankruptcy-throws-in-the-towel-jim-turley-electronics-engineering-journal/ Accessed 22.03.2023.

RISC-V. 2018. RISC-V SoftCPU Contest Highlights. https://riscv.org/blog/2018/12/risc-v-softcpu-contest-highlights/ Accessed 19.04.2023.

Sourceware. 2023. GDB information. https://www.sourceware.org/gdb/ Accessed 24.03.2023.

United Nations. The Paris Agreement. What is the Paris Agreement? https://unfccc.int/process-and-meetings/the-paris-agreement Accessed 15.05.2023.