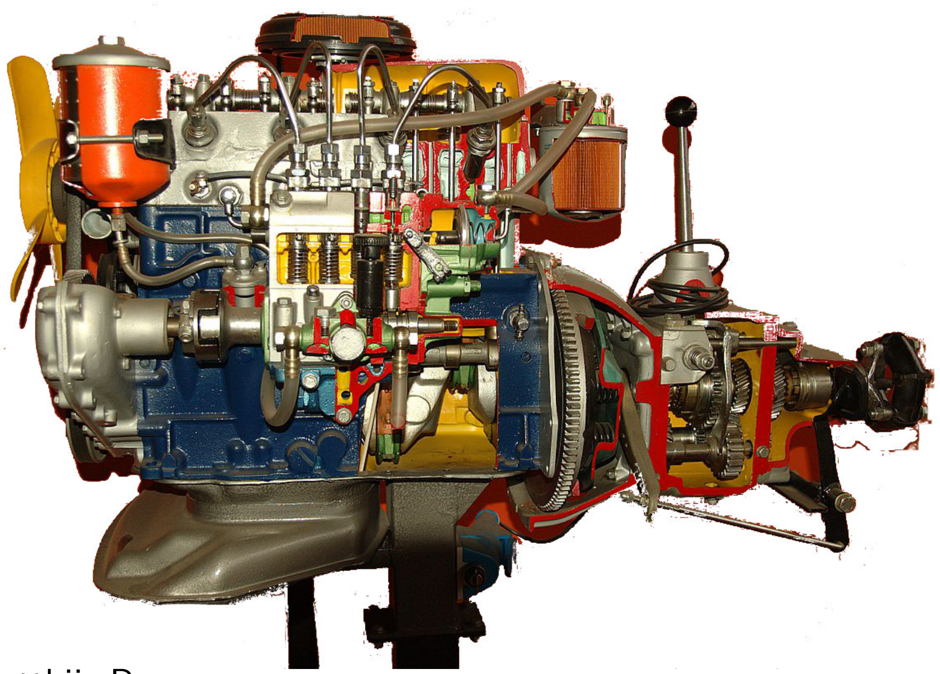
A model for experiment setups on FPGA development boards



Matthijs Bos

March 9, 2016

Supervisor(s): Anthony van Inge (UvA), Taco Walstra (UvA)

Signed: n.a.

Abstract

TODO

Contents

CHAPTER 1

Introduction

The use of field-programmable gate arrays (FPGA) has increased a lot in the past years. FPGAs have been successfully applied in many industries as well as in academic research. Manufacturers have put a great effort in further developing the capabilities of FPGAs, resulting in an increase in performance and size, as well as a decrease in power consumption and unit cost. Due to their high cost, FPGAs used to be unsuitable for application in classroom teaching. The developments of the past years however, put FPGAs within the reach of academic education. Nowadays, a full-featured FPGA development board can be acquired for under $150.

Computer architecture and organization is considered an important subject in the computer science program taught at the University of Amsterdam (UvA). The subject is taught to freshmen as an eight week course in an early stage of their curriculum. The subject's body of knowledge ranges from a low level understanding of a computer's central processing unit's (CPU) inner workings, design and surrounding systems to a more abstract view of the CPU that considers its instruction set from a software point of view.

The UvA's course contents are based on the widely adopted works of Hennessy and Patterson, supported by a number of lab experiments. These lab experiments allow students to reinforce their theoretical understanding of the course's subject matter by interacting with a number of virtual computer architectures in a simulator environment. In the process of modernizing the computer architecture and organization course, the instructors sense the need for a more hand-on experience in which students are capable of interacting with physical devices. This physical view of a computer architecture may aid in a student's understanding of the subject matter.

Hardware design is not part of the computer science curriculum.

Using FPGAs to experiment with (variations of) live computer architecture implementations to perform qualitative experiments instead of doing (partial) actual implementation. HDL programming is not a part of the curriculum.

FPGAs allow for a computer architecture design to be taken out of the virtual environment of a simulator and be implemented in a physical device. As such, they may provide a solution in achieving the goals of the UvA's computer architecture and organization course instructors. The technical capabilities of FPGAs combined with the decreased cost of FPGA development boards makes their adoption in the course a viable option.

Utilizing FPGAs however, is a complex task that requires a specific set of skills, including digital design, hardware definition language (HDL) programming and familiarity with specialized development tools. Their complexity and the set of skills required raises problems in the application of FPGAs in an eight week introductory level computer architecture and organization course. Each of these skills may deserve a course on their own and are mostly beyond the scope of the UvA's undergraduate computer science curriculum.

This thesis presents a generic solution that allows for students to utilize FPGAs in their lab experiments without the requirement of these more advanced skills. Experiments are performed using a Digilent Nexys 4 FPGA board, connected to a PC. An implementation is provided and validated through a number of experiments in which multiple variants of Hennessy and Patterson's MIPS computer architecture are running on an FPGA. The scope of this thesis is limited to a technical solution and does not evaluate the didactic effects of a hands-on approach in the teaching of these subjects.

1.1 Related work

Traditionally, computer architecture and organization lab experiments make use of virtual environments in the form of simulators. Many of these simulators exist, each with a different scope and level of detail. In particular, computer architecture simulators are used, which simulate the behaviour of a microprocessor. Some simulators such as the SPIM MIPS simulator [1] only feature a simulation of the instruction set and register state, while others provide a cycle accurate simulation of a specific microprocessor implementation, allowing for accurate evaluation of performance metrics. For their computer architecture and organization lab experiments, the UvA's course instructors makes use of the SIM-PL simulation environment [2], which was developed at the UvA. The SIM-PL simulator allows for a cycle-accurate simulation of the execution of a particular computer architecture that is implemented using digital design primitives through a graphical interface.

Jansen and Dusch [3] present a method that teaches electrical engineering master students to apply their available knowledge of HDL programming and digital electronics in order to implement and verify different parts of a microprocessor in VHDL. At the end of the course, students combine their previous results into a working microprocessor in order to produce a working system which is programmed onto a FPGA. After achieving a working setup on a FPGA, students are required to test their work by writing and running a simple c program. The course instructors provide the tools and a simple operating system in order to compile and run the student programs.

In the works of Cifredo-Chácon et al. [4], a teaching method is presented which provides first-year computer engineering students a more practical experience in the teachings of computer architecture and organization, as well as an introduction into the subject of VHDL programming. The lectures are combined with lab activities in which students perform experiments using VHDL and a FPGA development board. In their conclusions, the instructions observe the positive effects of a hands-on approach. As a weakness in their approach however, the instructors detect the difficulty students experience in adopting the parallel behaviour of VHDL as opposed to the familiar, imperative style of programming.

In their teaching of the subjects of digital design, Pereira et al. [5] make use of BIP, a simple computer computer architecture that is applied in both bachelor and master curricula. Bachelor students are introduced to the subject of digital circuit design through schematic implementation of parts of the computer architecture. Their work is validated in a simulator. In a digital systems design course, master students are given the task of providing a VHDL implementation of the computer architecture, validate their work through simulation and finally synthesize and run their work on a FPGA.

In order to provide students with a hand-on experience, El-Din and Krad [6] incorporate FPGAs in their computer architecture labs. Over the course of four lab sessions students are introduced into the use of FPGAs and their design software. Students are then required to model an ALU and verify its behaviour through simulation. A FPGA is then used to verify the ALU's behaviour in real life.

Holland et al. [7] present a solution in which a FPGA development board is used in collaboration with a student's pc in order to allow for debugging of internal microprocessor signals. A MIPS-based single-cycle as well as a pipelined microprocessor is programmed onto a FPGA development board and can be monitored and controlled through a graphical interface on the PC. A similar approach is presented in the works of Bulić et al. [8], which offers a pipelined
.

Xing et al. [9] describe a method in which FPGAs can be applied in order to perform experiments on a variety of multi-core system configurations.

1.2 Problem statement

In order to provide students with a hand-on approach, FPGAs have been applied in labs of computer architecture and organization in curricula of electrical enginering [3], [5], [6], [10], computer engineering [4]–[6], [10]–[12] and computer science [5], [6], [8], [12]. In electrical engineering curricula, the subject is taught as a means to provide students a top-down view [3]. Moving into the area of computer engineering and computer science, the subject serves as a means to reinforce a student's understanding of a computer's underlying hardware [4], [6], [12].

Labs are based on development boards and software from either Xilinx [8], [13]–[16] or Altera [3], [6], [9]–[12].

A more hand-on approach is required [4], [10]

FPGAs are used in a late stage of the course in order to let students' verify their work [14].

El-Din and Krad [6] recognise that computer science students lack experience with physical hardware.

Many have seen increased student scores as the result of the adoption of a hand-on experience in their labs.

Increased pass rate and score [16]

Increase in student evaluation [16]

incresed learning experience [10]. Increased interest [4]

Increased insight [4], [8]

Many educational programs assume HDL programming skills [3] or teach them as part of their curricula, while others teach aim to teach HDL programming as part of their courses. Cifredo-Chacoón et al. [4] observe that first-year students struggle in the adoption of HDL programming concepts.

Third-year undergraduate course on programmable logic [11]

FPGAs and VHDL are taught to master students [5]

VHDL and FPGAs taught to third-year undergraduate students already familiar with the topics of digital design and microcontroller design [11]

VHDL and FPGAs as part of their masters level education [3], [5]

The subject of computer architecture and organization is taught different to electrical engineering students as opposed to computer science/computer engineering students [17]. OLD SOURCE.

Students that have not experimented with more abstract subjects such as pipelining and caching show lower test scores [16]

Approach focused around a specific implementation of an instruction set.

Many educational computer architectures have been developed for this purpose. The instruction set is often reduced of simplified.

Approach where students build their own CPU throughout the course. The focus is on implementation and not on experiments with these implemented computer architectures. More low level approach, not so much a high level approach in which students are encouraged to evaluate the effects of high-level strategical decisions.

VHDL programming exercises are limited to simple exercises such as implementing an ALU [6] or implementing a very limited microprocessor **nativeFPGA**

FPGAs are not suitable for an experimental way to learn VHDL, since is requires at least some minutes to synthesize, implement and program a design every time a change is made to the code. Simulators are more suited for this purpose.

FPGA applied more in electrical engineering or computer engineering curricula.

* to HDL compilers

Many others develop their own instruction set/computer architecture that is not compatible with H&P.

1.3 Problem Statement and Related Work

FPGA's have seen successful application in a number of fields, ranging from industry to academic research. Education has been one of these fields as well, for a number of the FPGA development board is not very well suited for application in classrooms where it serves as a tool to aid the student in its understanding of the subject matter of digital circuits and systems. This is mainly due to the fact that FPGAs are complex devices and most of current methods require the student to acquire or have knowledge on the subjects of FPGA development and HDL modelling. Furthermore, there currently exist no tools that aid instructors in the development and sharing of experimental setups for use in their lectures.

Most existing methods that incorporate FPGA development boards in their experiments require students to familiarize themselves with the FPGA development process, as well as HDL programming. These topics however, are often beyond the scope of curricula of Computer Science and Software Engineering, putting these methods outside the reach of many students. Exisiting methods often consider FPGA development to be a primary educational goal, in stead of being a means to teach other subjects. By requiring students to acquire knowledge on these topics, a lot of available time is consumed by matters other than the actual goal.

HDL programming concepts are often misunderstood by students participating in entry level courses. HDL modelling and FPGAs are often considered an advanced topic, taught in a later stage of the curriculum.

1.4 Problem Statement 3.0

beschrijf het probleem: er zijn op dit moment geen methoden die eerstejaars studenten in staat stelt door middel van FPGAs een fysieke indruk te krijgen van de hardware waar zij mee experimenteren in practica. Dit komt in ieder geval niet naar voren in de volgende papers....

De oplossing hiervoor is volgens de docenten het volgende: Men wil een technische oplossing die studenten zonder technische kennis van FPGA's en HDL programmeren in staat stelt FPGA's te gebruiken tijdens practica. Deze vaardigheden vallen immers buiten het curriculum van de informaticaopleiding. Bovendien weet ik uit eigen ervaring hoe moeilijk het is om FPGA en HDL programmeren onder de knie te krijgen. Experimenten moeten snel en eenvoudig uitwisselbaar zijn alsof de FPGA alleen maar een container is die de logica inbed. Veel methoden zien de beheersing van het ontwikkelproces voor FPGA's als een onderwijsdoel, maar dat is volgens de docenten niet de bedoeling. Het gaat heb meer om de observeerbaarheid van complexe opstellingen en de interactie om vervolgens een diepgaande analyse te kunnen doen van de resultaten. De huidige methoden beperken zich toch vaak tot relatief eenvoudige experimenten en richten zich meer op implementatie.

Verder vinden ze FPGA's een geschikt middel, dit wordt ondersteund door de volgende papers:.... Bovendien maakt een hardwarecomponent het onderwijs aantrekkelijker en uit literatuur blijkt dat het inzetten van FPGA's leidt tot verbeterde leerprestaties. FPGA's zijn dus in de basis een geschikt middel voor onderwijs: ze zijn goedkoop, flexibel inzetbaar. Ook beter dan simulaties, omdat je een hands-on ervaring hebt, wat volgens de docenten beter is dan een virtuele oplossing. I.t.t. ASICs zijn FPGAs flexibel en zouden de ontwikkelkosten van een practicum veel hoger liggen.

Voortbordurend op wat de docenten als oplossing voor zich zien, zie ik verder het volgende: Het ontwikkelproces is erg omslachtig en vereist veel kennis. Bovendien zijn er geen standaardcomopnenten en zijn er momenteel geen methoden die het werk verdeeld volgens specialisme. Niet alleen het experimenteren vereist volgens de huidige methoden veel kennis van FPGA's maar ook het ontwikkelproces vereist onnodig veel vaardigheden. Fundamenteler: de huidige modellen die bestaan voor het gebruik van FPGA's zijn ongeschikt voor onderwijs. Om de bovengenoemde problemen op te lossen moet er eerst nagedacht worden over een model dat zowel studenten als docenten in staat stelt effectief FPGA's in te zetten als gereedschap voor experimenten. Ook moet dit model een oplossing bieden voor het gebrek aan uitwisselbaarheid van experimenten. Docenten ontwikkelen nu wel wat, maar hun werk is totaal niet gestandaardiseerd. Het meeste werk is specifiek gericht op een bepaald stuk hardware en soms is er een specifiek stukje copmutersoftware bijgeschreven. Door het gebrek aan standaarden is het ook moeilijk om experiment opstellingen op een juiste manier te kunnen delen met anderen, terwijl ontwikkeling voor FPGAs juist een grote investering vereist op het gebied van

1.5 Research Question

In order for FPGAs to provide students with a hands-on experience during their lab experiments, a solution is required that removes the need for skills such as HDL programming and abstracts the complexities of working with FPGAs.

This thesis' research question is as follows:

> How can FPGAs be applied in computer architecture and organization lab experiments, while hiding their technical complexities and removing the need for HDL programming skills?

CHAPTER 2

Background

2.1 Field-programmable gate arrays

Field-programmable gate arrays (FPGA) are a class of integrated circuits (IC) which' behaviour is reconfigurable, as opposed to application specific integrated circuits (ASIC). In their most common form, FPGAs consist of configurable logic blocks (CLB), combined with a configurable network that interconnects these. Many manufacturers combine CLBs with additional static functionality such as memories, I/O controllers and even embedded microprocessors. Which static features are included differs per FPGA model and manufacturer. Figure 2.1 gives a structural overview of a FPGA as manufactured by Xilinx. It shows a number of CLBs, interconnected with I/O blocks (IOB) that connect IC package pins and block rams (BRAM) which serve as static blocks of memory. The digital clock manager (DCM) is responsible for the distribution of clock signals to every of the FPGA's blocks.

A CLB's contained logic is described through an internal lookup table in which

2.1.1 I/O capabilities

A FPGA development board generally includes a number of I/O devices. Simple devices for input include buttons and switches, while simple output devices are generally LEDs or LED displays. Furthermore, a development board often includes a set of pin headers, which can be configured individually to be input or output ports.

More complex I/O devices are included as well. Such as VGA output controllers, microphone,

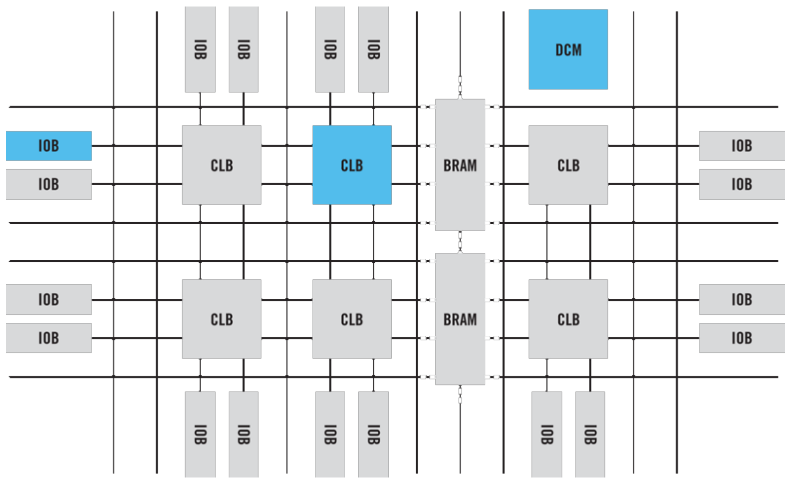


Figure 2.1: Structural FPGA overview [18]

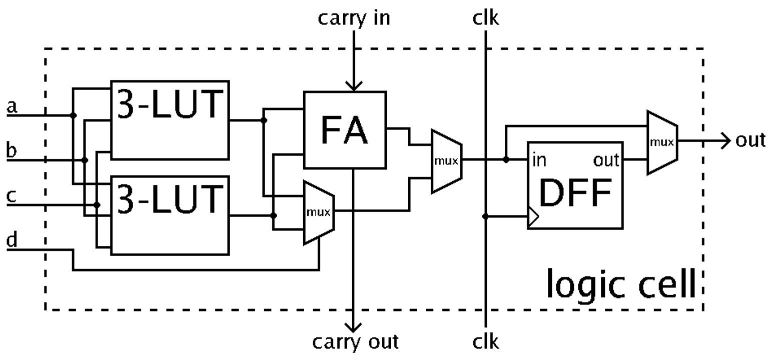


Figure 2.2: FPGA CLB example overview [19]

Ethernet, USB, ... These devices operate within strict timing constraints in order to work properly. This prevents the experiment setup from properly using the device, since it runs on a irregular, low frequency.

Buffering (right term?) of some types of input, such as push buttons. When a user presses a push button while the experiment is halted, the signal should be artificially set to be high during the next cycle.

I/O devices can only be seen as combinational circuits. I/O devices that are based on sequential logic, such as a LED display controller, an abstraction needs to be created such that the device behaves like a combinational circuit.

2.1.2 Platform dependence

Earlier approaches that have applied FPGAs in education have resulted in solutions that are very board-specific. FPGAs manufactured by both Xilinx and Altera have been allied in the education of these topics. Both manufacturers have tightly integrated their software development kits with their FPGAs. Development board are manufactured by these manufacturers, as well as third parties, such as Digilent.

Hardware design projects may be interchangeable between different versions of a FPGA chip, as well as different product families.

2.1.3 Programming

Current FPGAs generally make use of the IEEE 1532 standard for in-system programming, which extends the IEEE 1149.1 'JTAG' boundary scan protocol. In order to program these FPGAs, one makes use of dedicated JTAG programming hardware and software. Such hardware and software is commonly available.

FPGA development board manufacturers generally integrate JTAG programming hardware in their designs, allowing progamming over a serial connection to their PC, such as USB. Manufacturers combine their development boards with software development kits that facilitate the operating system drivers and software required for the programming process. Other means of programming are seen as well, such as support for USB storage devices or memory cards on which a file is placed that is automatically read by the development board and programmed onto the FPGA. FPGA development boards generally feature a JTAG interface as well, allowing for a custom method of programming using external tools.

2.1.4 PC Communication

In order to allow for communcation with a PC, FPGA development board manufacturers generally include one or more communication interfaces in their board designs. Many FPGA development boards currently available come with a USB interface over which a virtual serial interface is exposed, commonly known as a virtual COM port. The serial signal is often controlled by a UART implementation that is programmed onto the FPGA. Other examples of communcation interfaces are RS232 and Eternet ports, where dedicated hardware components on the FPGA development board are responsible for translating signal levels controlled in the FPGA to interface standard levels.

TODO uitleggen waarom synchronous sequential logic niet werkt op FPGAs. Timing sensitive, zeer geavanceerde place & route. Wordt niet ondersteund door tools.

CHAPTER 3

Model

This chapter will propose a model that provides a theoretical solution to the problem statement as described in section 1.2. This model addresses the problem statement from a technical and architectural point of view, as well as how user processes are affected. As a starting point, a description of a basic model for experiment setups on FPGA development boards is given in section 3.1, which is then extended and modified through a series of stages. In every stage, a particular aspect of the problem statement is addressed, forming a complete solution eventually.

In section 3.2 the possible limiting factor of the number of I/O devices on the FPGA development board is addressed by introducing the concept of a controller. This controller provides the experiment setup logic with a variable amount of virtual I/O channels. The levels of these I/O channels can then observed and controlled through controller-specific PC software. All of the physical I/O devices are temporarily removed from the model, as well as most of the FPGA's other peripheral devices.

In section 3.3 the controller is extended such that it allows for cycle-accurate control of sequential experiment setup logic. Users control cycles through the PC software, but the model supports autonomous operation as well. This new functionality is added through expansion of the experiment setup interface.

In section 3.4 the experiment controller's interface is generalized, such that the interface dependencies between experiment setup and controller are removed. This allows for independent development of controllers and experiment setups, allowing for random combinations of board-specific controllers and independent experiment setup logic. The controller's PC interface is generalized as well, which allows for the development of universal PC software.

In section 3.5 the FPGA development board's physical I/O devices are reintroduced into the model, after being removed from the model in section 3.2. This step is essential in enabling a student's physical view of the experiment setup and providing a hands-on experience. The development of experiment setups remains an independent process.

3.1 The Basic Model

As a starting point, a description of a basic model for experiment setups on FPGA development boards is given. An overview of the basic model is displayed in figure 3.1. The model features a PC and a FPGA development board as the two primary physical components. An interface between the PC and the FPGA development board exists over which the board exposes the `Board.Program()` operation. This operation initializes the FPGA by loading the contents of a FPGA-specific bitstream file and configuring the FPGA's components. The physical and electrical characteristics of this interface are considered to be irrelevant and may even be a process that involves manual user operations, such as transferring a memory card.

In this basic model, the FPGA development board is considered to be a printed circuit board (PCB) which hosts and interconnects a FPGA and its various peripheral components. Not all pins of the FPGA package and not all peripheral devices are considered to be relevant to the model.

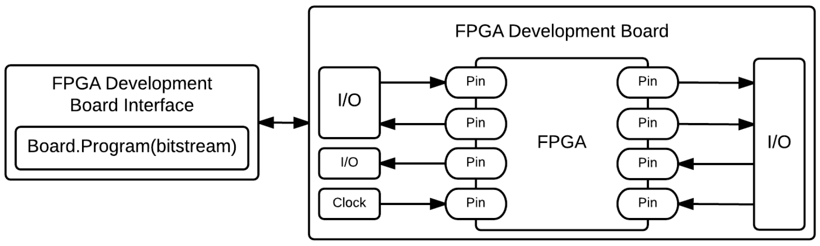


Figure 3.1: The basic model, an overview of the FPGA development board and its exposed interfaces.

Only the pins whose signals can be controlled through the FPGA's contained logic are included. Peripheral devices that do not connect to these pins, such as power supplies or programming circuits are excluded from the model. Specifically, the presence of a clock generating device is assumed, providing the FPGA with a clock signal on one of its pins. Although this model is a great simplification of reality, this abstract view is assumed to be sufficient for the purposes of this model and applicable to most FPGA development boards.

Besides the omission of details of the FPGA's peripherals, a part of the FPGA's internal complexities are hidden from the model as well. Figure 3.2 gives a graphical overview of the FPGA's internals and its contained logic. The FPGA's internal interface is simplified and defined to be a container for the end product of a HDL developer's work: an entity with input signals, output signals and an input clock signal. Other physical, electrical or logical characteristics of the FPGA are not included in the model.

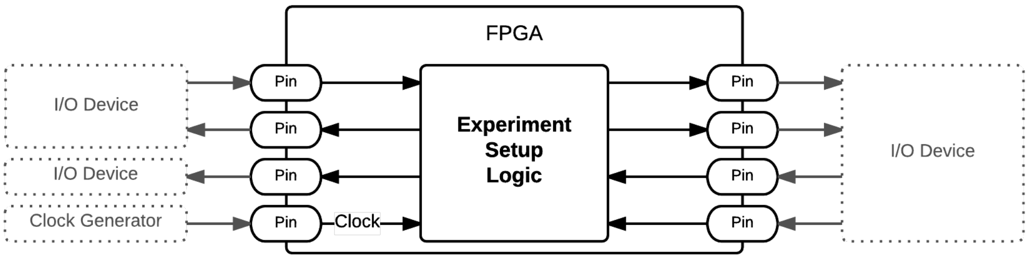


Figure 3.2: The basic model, an overview of the FPGA and its contained experiment setup logic. No specific architecture is applied to the logic.

In the case of this basic model, no specific architecture is defined that will embed the experiment setup logic into the FPGA development board. A specific architecture will be developed in the following stages. At this stage of the model, a developer is responsible for the development of its own architecture that embeds the experiment setup logic into the environment of the FPGA.

In this basic model, two user roles are defined: experiment setup developers and experimenters. In a classroom environment, an instructor can be considered an experiment setup developer and a student can be considered an experimenter.

3.1.1 Experiment Setup Developers

Experiment setup developers are responsible for the design, implementation, testing, documentation and distribution of experiment setups for use on FPGA development boards. The end product of their work is a software package that contains a bitstream file and optionally the source files used in the bitstream file's compilation process. Figure 3.3 displays a graphical overview of the experiment setup development process.

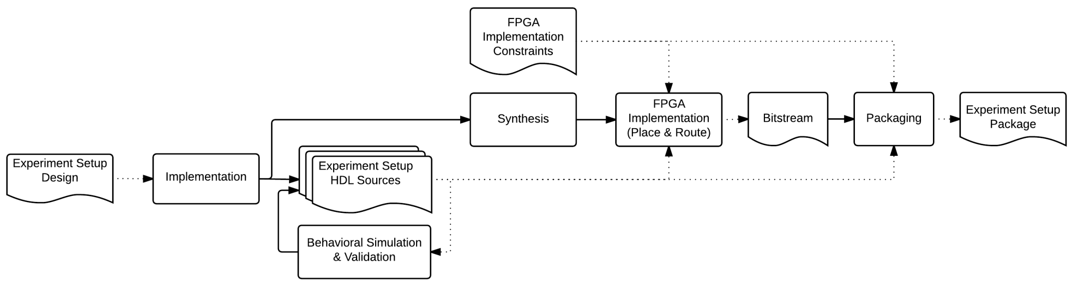

FPGA Implementation Constraints
Experiment Setup Design
Implementation
Synthesis
Experiment Setup HDL Sources
Behavioral Simulation & Validation
FPGA Implementation (Place & Route)
Bitstream
Packaging
Experiment Setup Package

Figure 3.3: The basic model, an overview of the experiment setup development process.

As shown below, an experiment setup developer must have knowledge of a significant number of subjects and technologies:

- Designing an experiment setup requires knowledge on the subject of digital logic and digital systems.
- For the design to contain meaningful educational content, the developer should have experience in teaching the subject.
- Translating the design into a working, valid implementation in VHDL or Verilog requires experience in HDL development.
- Embedding the implemented design in a FPGA development board requires knowledge of the specific FPGA, its development tools and the FPGA's peripheral devices.

3.1.2 Experimenters

Experimenters are responsible for carrying out experiments. They obtain experiment setup software packages and initialize experiment setups on their FPGA development boards. In order to complete the experiment, they make observations and interact with the experiment setup that is contained within the FPGA. Figure 3.4 displays a graphical overview of the experimentation process. In this basic model, two different methods of interaction are defined: interaction through board I/O devices and interaction through the process of HDL source modification, recompilation and reprogramming. Observation of the experiment's results can be done through the FPGA development board's I/O devices. Some development tools allow for live inspection of the FPGA's internal signal levels through a PC connection.

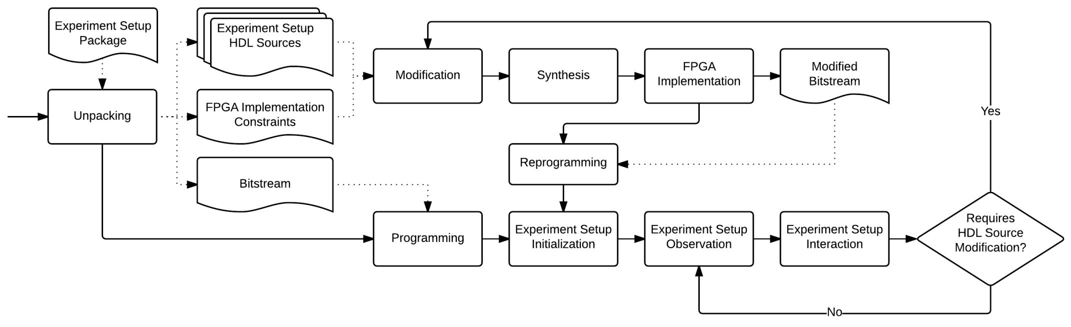

Experiment Setup Package
Unpacking
Experiment Setup HDL Sources
FPGA Implementation Constraints
Bitstream
Modification
Synthesis
FPGA Implementation
Modified Bitstream
Reprogramming
Programming
Experiment Setup Initialization
Experiment Setup Observation
Experiment Setup Interaction
Requires HDL Source Modification?
Yes
No

Figure 3.4: The basic model, an overview of the experimentation process.

In order for a user to act as an experimenter, one must understand the logic and workings of the obtained experiment setup. Furthermore, an experimenter must understand the basic concepts and role of the FPGA development board, as well as how to program the FPGA via the experimenter's PC. In preparation of the experiment, experimenters must install programming software and operating system drivers for the FPGA development board. In this case, interaction and observation is done through the board's I/O devices.

Including the PC as a tool for observation and interaction requires additional knowledge and preparation from experimenters. In order to understand and modify the experiment setup's HDL sources, users must be familiar with the programming language used. Recompilation requires users to install and understand the FPGA's development tools in order to set up a proper development environment. Furthermore, this method of interaction will significantly increase the time required for every change to be processed, since the process of recompilation is a slow process. Using the FPGA as a tool for observation requires further familiarization with the FPGA's development tools.

3.2 Virtualizing I/O

Regular FPGA development boards offer a limited set of I/O devices. As a consequence, this allows for observation and control of experiment setup logic entities with a limited number of input and output signals. Embedding entities with a large number of inputs and outputs however, requires a different approach.

In order to support experiment setup entities with a large number of input and output signals, the basic model's logic architecture is extended through introduction of the controller. Figure 3.5 gives an overview of the FPGA and the newly defined architecture. The experiment setup entity's inputs and output signals are available to the controller only, but both components still share a common clock signal. The signal levels of the FPGA's peripheral devices are no longer driven by the experiment setup entity. Following the previous section's criteria, these devices are considered irrelevant and thus removed from the model. Only the clock generating device and a single machine-machine communication device remain part of the model.

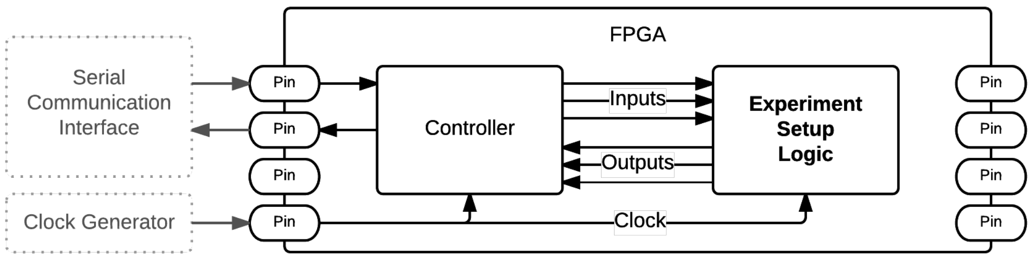


Figure 3.5: I/O virtualized, an overview of the FPGA and its contained logic architecture. The controller embeds the experiment setup logic into the FPGA.

The controller is a component that acts as an intermediary between the experiment setup logic and an experimenter's PC. The experiment setup's state is observed and controlled through controller-specific software, a method also presented in [7] and [8]. A hardware-based method to observe and control the experiment setup's state is described in [13]. A software-based solution however, does not increase the cost of the experiment. Furthermore, a software-based approach will allow for more flexibility, as shown in the following sections.

The controller exposes an additional interface through a communication channel that must be provided by one of the FPGA development board's communication devices, as can be seen in figure 3.6. This interface defines operations that allow for the experiment setup entity's individual input and output signals to be controlled and observed respectively. A similar interface for observation and control of specific signals is described in [7].

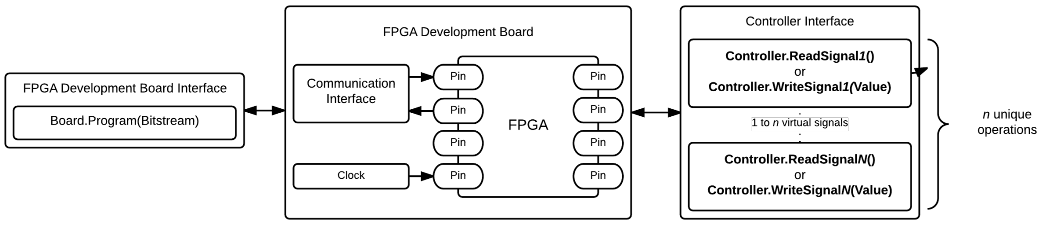


Figure 3.6: I/O virtualized, an overview of the FPGA development board and its exposed interfaces.

The explicit separation of concerns in the logic architecture allows for separate implementation and validation processes for experiment setup logic, controller logic and PC software. This partitioning subsequently allows for distribution of work among specialists, removing the need for a single experiment developer to have knowledge and experience in all the areas previously described in section 3.1.1. As a consequence of the separation however, more dependencies are introduced into the development process. The development of a controller component requires a definition of the experiment setup's interface and any change in this interface definition requires modification of the controller. A similar dependency exists between the controller and the PC software. A change in the experiment setup entity's interface definition will thus not only result in the need for modification of the controller, but in the need for modification of the PC software as well. These dependencies are addressed and removed from the model in section 3.4.

Experiment setup interaction through dedicated PC software will simplify the experimentation process for experiment setups with large number of input and output signals. Involving the PC removes the need for experimenters to install and familiarize themselves with the FPGA's development tools. Experimenters will be able to interact with the experiment setup through a graphical interface on the PC, removing the need need for HDL programming skills and allowing for real-time interaction, since no recompilation is required. Since the board's I/O devices have been removed at this stage of the model, there is no possibility for users to physically interact with the experiment. The PC software provides the only means of interaction with the experiment setup. These removed I/O devices however, will be reintroduced into the model in section 3.5.

3.3 Cycle control

In the previous section, the basic model has been extended to include a definition of the controller. This controller allows for the embedding of experiment setup entities with an arbitrary number of input and output signals that are defined through combinational logic. FPGAs in general lack practically useful support for containment of asynchronous sequential logic, but the availability of a clock signal in the current model does provide support for experiment setup entities that are defined through synchronous sequential logic. However, this clock signal is constant and cannot be varied in speed or temporary stopped. Embedding an experiment setup entity of synchronous sequential logic at this stage of the model would result in an uncontrollable situation of continuous state changes at high speed. This limited control over the clock signal is due to the nature of how FPGAs work and prevents cycle-accurate interaction with the experiment setup. In order to be capable of cycle-accurate observations and control over synchronous sequential experiment setup logic, the model requires further extension.

As a solution to the problem stated above, the FPGA's contained logic architecture has been extended, as displayed in figure 3.7. The most significant change is the addition of the `clock_enable` input signal on the experiment setup's interface definition. Combined with the `reset` signal, these new input signals allow for cycle-accurate control and reinitialization of the experiment setup logic's state. The introduction of the `clock_enable` signal will require modification of every synchronous memory element present in the experiment setup logic, as

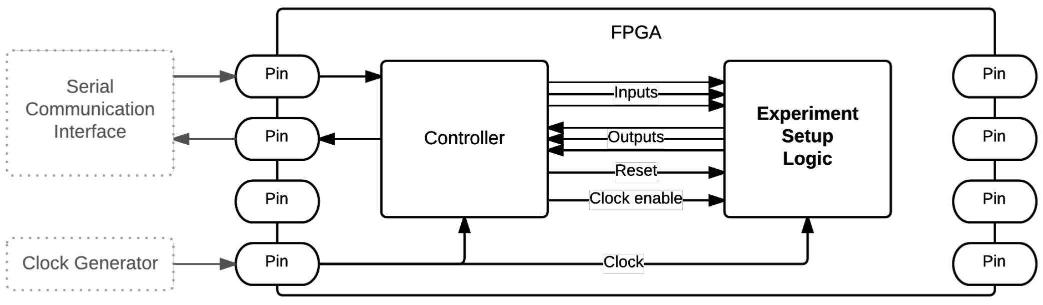
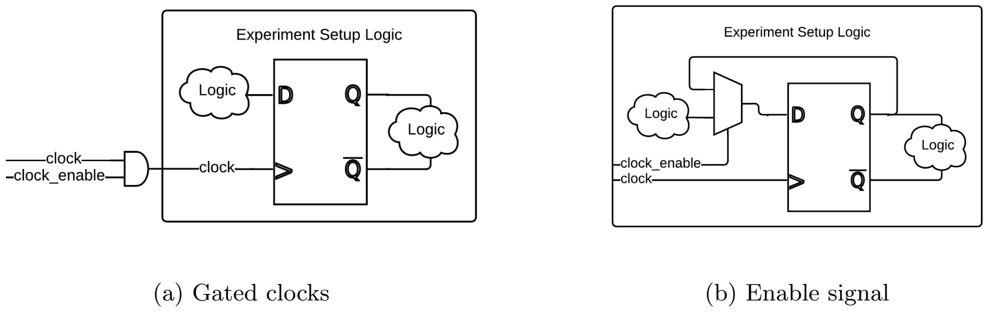


Figure 3.7: Cycle control, an overview of the FPGA and its contained logic architecture. The `clock_enable` and `reset` signals are added to allow for cycle-accurate interaction and (re)initialization.

displayed in figure 3.8b. By multiplexing the input signal for every memory element based on the same `clock_enable`, one can control the state change in a cycle-accurate manner. The method of using enable signals to control synchronous logic is described in [20, Sec 2.4.5].

One may argue that this approach is unfavourable, since it requires modification of existing experiment setup logic designs. This method for controlling state change however, is a common approach in FPGA-targeted HDL development and is used in many designs. More specifically, this approach was also taken in controlling the experiment setups described in [7] and [8]. As stated before, FPGAs do not allow for dynamic manipulation of clock signals by design, a method known as clock gating (figure 3.8a). Clock signals are distributed over the FPGA through dedicated lines that cannot be altered through HDL logic. Some manufacturers include support for static clock division, but still only down to relatively high speeds and do not allow for dynamic adjustment of clock speed [TODO: Bron].



(a) Gated clocks

(b) Enable signal

Figure 3.8: Controlling state transitions through a clock enable signal and a gated clock signal. FPGA's do not support gated clock signals.

In order to make this newly developed functionality available to the experimenter through the PC software, the controller interface is extended to support new operations that allow for the control of cycles. As displayed in figure 3.9, five new operations are available through the controller's interface. The `Controller.Reset()` operation (re)initializes the experiment setup to it initial state and waits for the next operation. Experimenters have manual control over the experiment setup's cycles through the `Controller.Step()` operation. The controller may also be instructed to manage the experiment setup autonomously through the `Controller.Start()` operation. The maximum operating frequency may be defined through the `Controller.LimitSpeed()` operation. The `Controller.Stop()` operation stops the controller's autonomous management of the experiment setup and awaits the next operation. Similar interfaces for cycle-accurate control of experiment setups are described in [7] and [8].

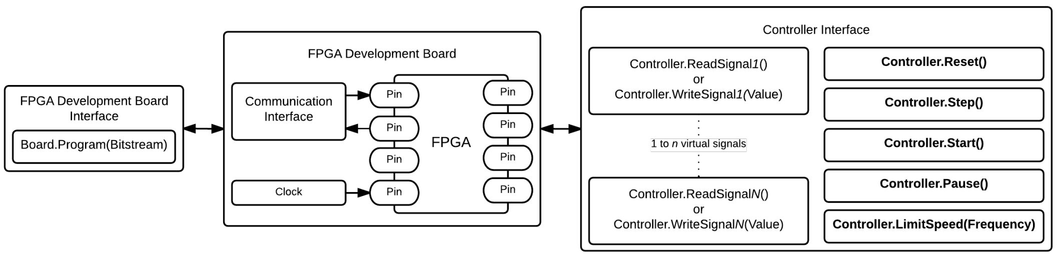


Figure 3.9: Cycle control, an overview of the FPGA development board. The controller's interface is extended with operations that allow for cycle-accurate interaction with the experiment setup.

3.4 Board Component Abstraction

While the model has developed to address a number of problems encountered by experimenters, the problems experienced during development have only been addressed partially. Although an architecture for the FPGA's contained logic was developed in the previous sections, a series of dependencies can still be identified in the development process, as displayed in figure A.3.

The source of these dependencies is the interface between controller and experiment setup. This interface is specific for every experiment setup. By defining a standard interface between experiment setup and controller, this dependency is removed. In the current approach, the experiment setup is considered an entity with inputs and outputs, supplemented by a clock signal, a clock_enable signal and a reset signal.

3.4.1 Address Space

In order to be able to provide a generic interface between controller and experiment setup logic, the experiment setup's interface is modified such that it resembles the interface of a block ram, as can be seen in figure 3.10b. The experiment setup's signals are projected on the address space.

Losing the information about which signal is which is irrelevant to proper operation of the controller. In order for the PC software to be of any practical use however, a definition of this memory space is required, such that the individual signals can be identified from the memory space.

The controller does no longer provide an interface to the experiment setup, but acts as a proxy to control the experiment setup's interface. This does not only allow for a reuse of controlling logic, but a simplification as well, since the interface that is exposed to the controller is simplified as well.

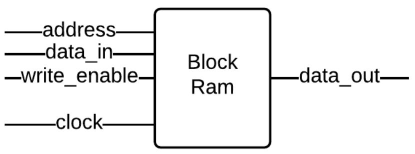
3.4.2 Interface

3.4.3 Memory devices

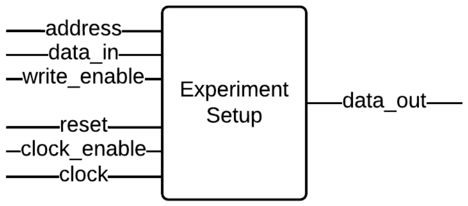
The experiment setup has been considered to be an entity with input output signals. Any memory elements contained within the experiment setup can be made accessible through these input and output signals. This solution is not very scalable however. In order to allow for the controller to read and write to large memory elements such as block rams and register files, a different solution is required. A solution to this problem is displayed in figure B.2.

A simple wrapper can be computer-generated in many cases. If complex memory devices are contained within the experiment setup, a wrapper can be defined manually, using an experiment setup adapter component, address space splitters and memory interface multiplexers.

Large memory elements such as register files and block rams may be defined in HDL. During the compilation process, these elements are infered from HDL code and the FPGA's internal block rams are used.



(a) Single-port block ram, derived from [21]



(b) Experiment setup interface

Figure 3.10: Interfaces

3.4.4 Composition Process

In stead of the distinction between controller and experiment setup, a new separation is defined in the controller's logic architecture. A distinction is made between the logic that is targeted to a specific FPGA development board and the logic that targets the standard board component interface. Figure 3.11 displays this distinction. This distinction is made, because the experiment setup is no longer being defined by it's primary logic, but as secondary interfacing logic as well.

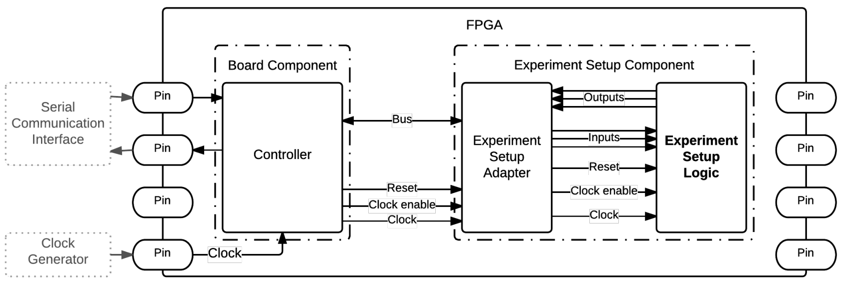


Figure 3.11: Controller abstracted, FPGA logic design

A new process is defined for the composition of board component and experiment setup component into a bitstream file. This results in the processes of experiment setup development and board component development to become independent. Figure A.4 displays the composition process and its relation to the development processes for experiment setup component and board development component in terms of dependencies.

3.4.5 Experiment Setup Adapter

In order for an experiment setup logic to conform to the experiment setup component's interface, a component is introduced that acts as an adapter. This experiment setup adapter is a logic component that projects the experiment setup's input and output signals onto an address space. Figure 3.11 displays the experiment setup adapter.

3.4.6 Generic PC Software

A major advantage of the generic interface between the controller and experiment setup is that PC software can be developed that is no longer specifically tied to one particular experiment setup design. The controller now acts as a proxy between the PC and the experiment setup. The experiment setup is considered an address space from the PC software, no longer an entity with inputs and outputs. In order for this address space to be of any significant meaning, the PC software requires information on how the data in this address space is structured. During the wrapping process a file will be generated, describing how the experiment setup's signals are projected onto its address space.

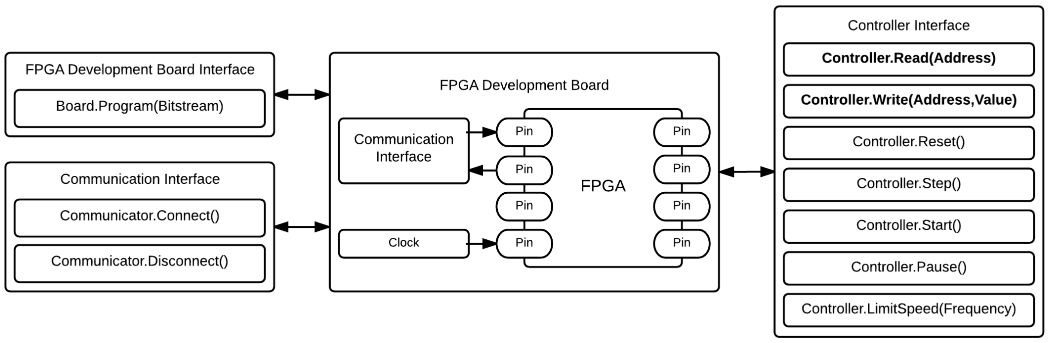


Figure 3.12: Controller abstracted, overview

3.5 Reintroducing Board I/O

Experiment setup has no accurate notion of time, so I/O devices can only be addressed as combinational devices. I/O device drivers facilitate this.

Board I/O capabilities are of vital iportance in providing a physical hands-on experience. Although the FPGA itself is a physical object, the board's I/O capabilities expose the hardware setup's physical behaviour and provide real life interaction capabilities. Without this step, the whole point of a physical hands-on experience is lost.

See figure 3.14

3.5.1 I/O processing

Optionally, one needs to introduce device driver logic in order to allow for sequential I/O device logic to be usable as combinational logic.

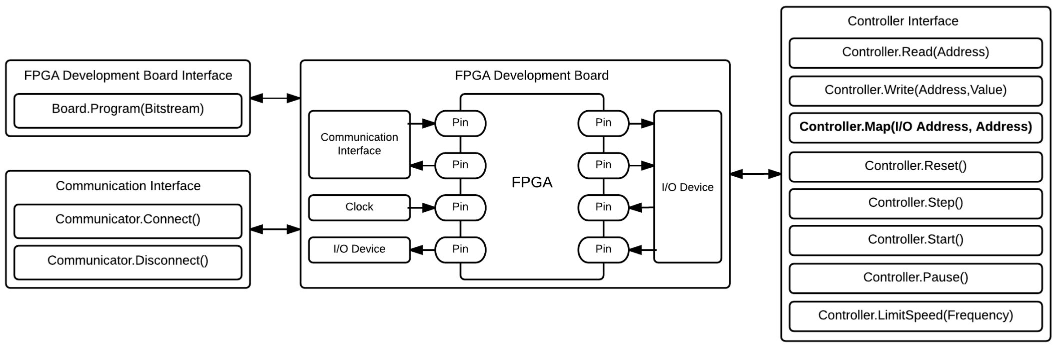


Figure 3.13: I/O reintroduced, overview

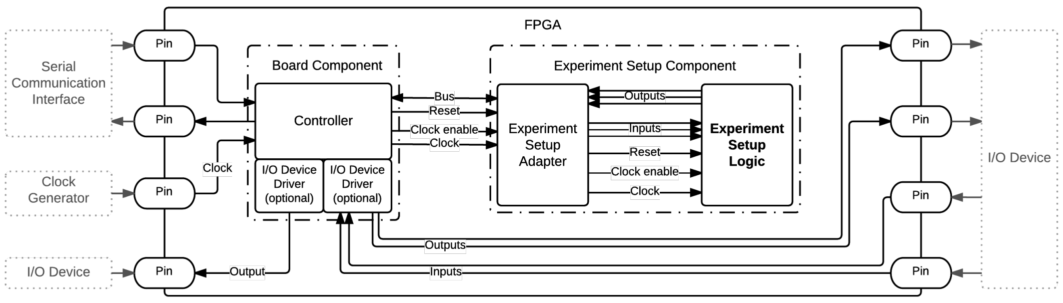


Figure 3.14: I/O reintroduced, FPGA logic design

CHAPTER 4

Implementation

In order to test the proposed framework, an implementation is provided. The framework implementation is targeted at the Digilent Nexys 4 FPGA development board, displayed in figure 4.1. This board contains a Xilinx Artix-7 series FPGA and is equipped with various I/O devices. The development board is connected with the PC through USB. This USB connection is used in order to setup a serial communication channel between the PC and the FPGA.

- PC Software Interface Standardization
- PC Software Development
- FPGA Development Board PC Software Plugin Development
  - For a specific FPGA Development Board PC Driver
- Board Package Interface Standardization
- Board Package Development
  - For a specific FPGA Development Board
- Experiment Wrapping Toolchain Development
- Composition Toolchain Development
  - For a specific FPGA Compilation Toolchain
- Experiment Setup Development

4.1 Functional

Reset button behaviour.

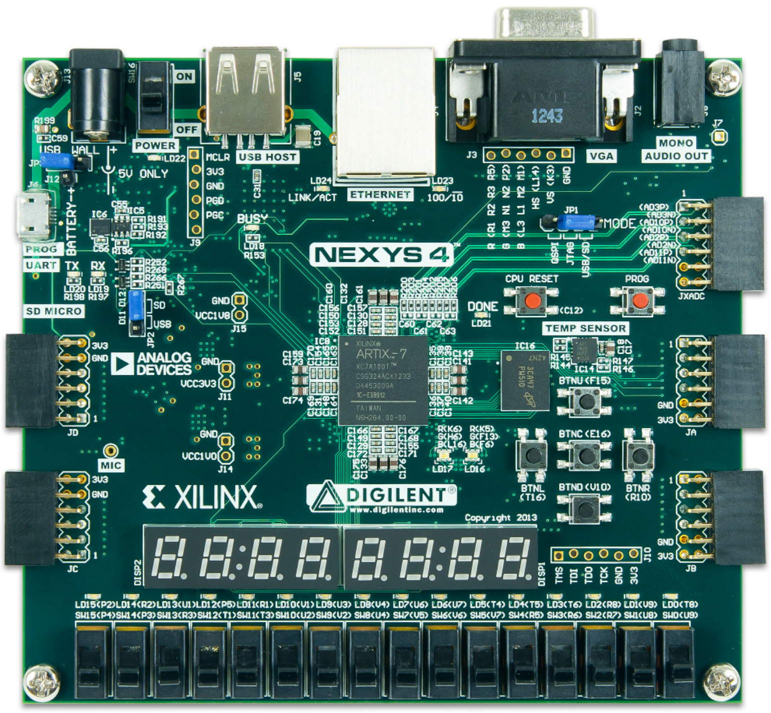


Figure 4.1: Digilent Nexys 4 Artix-7 FPGA development board

CHAPTER 5

Experimental results

APPENDIX A

Process Analysis

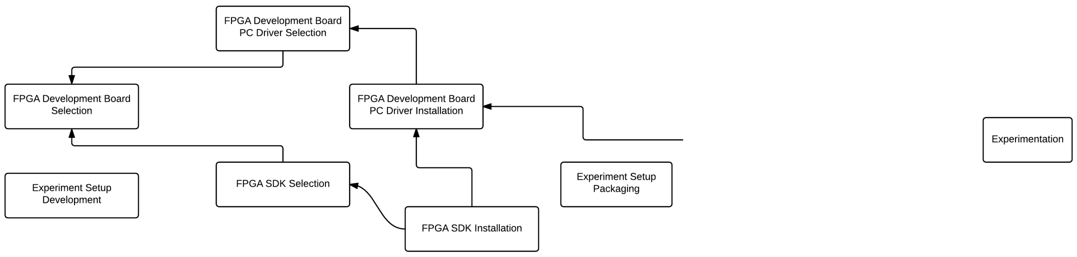

FPGA Development Board PC Driver Selection
FPGA Development Board Selection
FPGA Development Board PC Driver Installation
Experimentation
Experiment Setup Development
FPGA SDK Selection
Experiment Setup Packaging
FPGA SDK Installation

Figure A.1: The basic model, process dependency graph

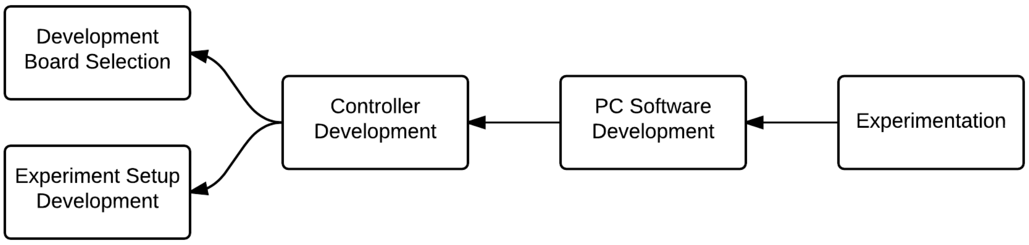


Figure A.2: I/O virtualized, process dependency graph

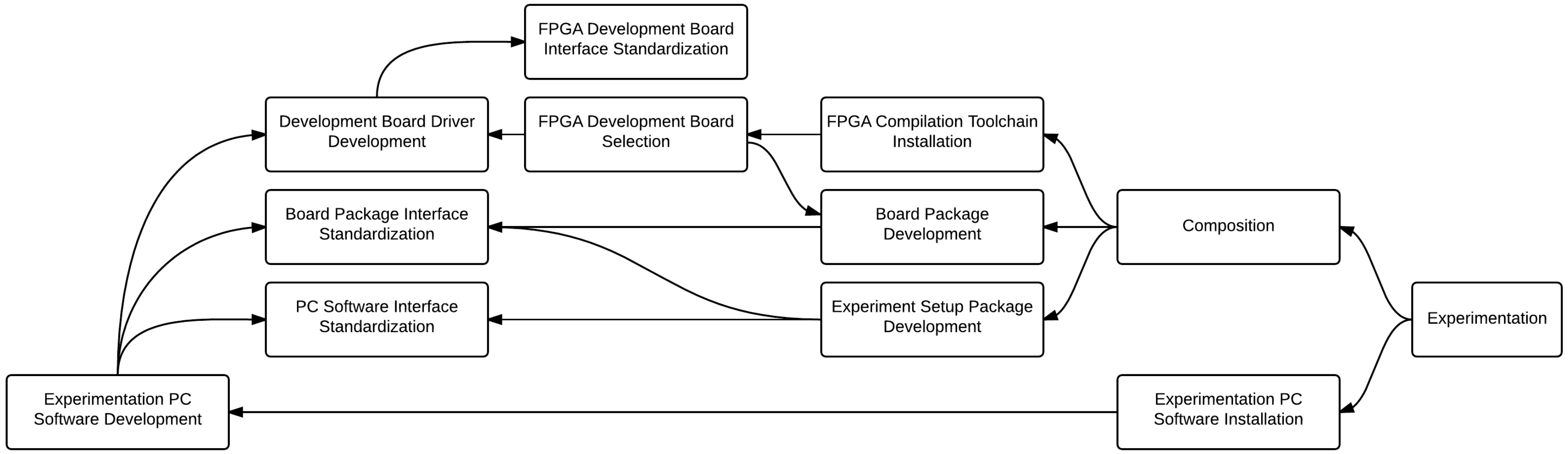

FPGA Development Board Interface Standardization
Development Board Driver Development
FPGA Development Board Selection
FPGA Compilation Toolchain Installation
Board Package Interface Standardization
Board Package Development
Composition
PC Software Interface Standardization
Experiment Setup Package Development
Experimentation
Experimentation PC Software Development
Experimentation PC Software Installation

Figure A.3: Cycle control, process dependency graph

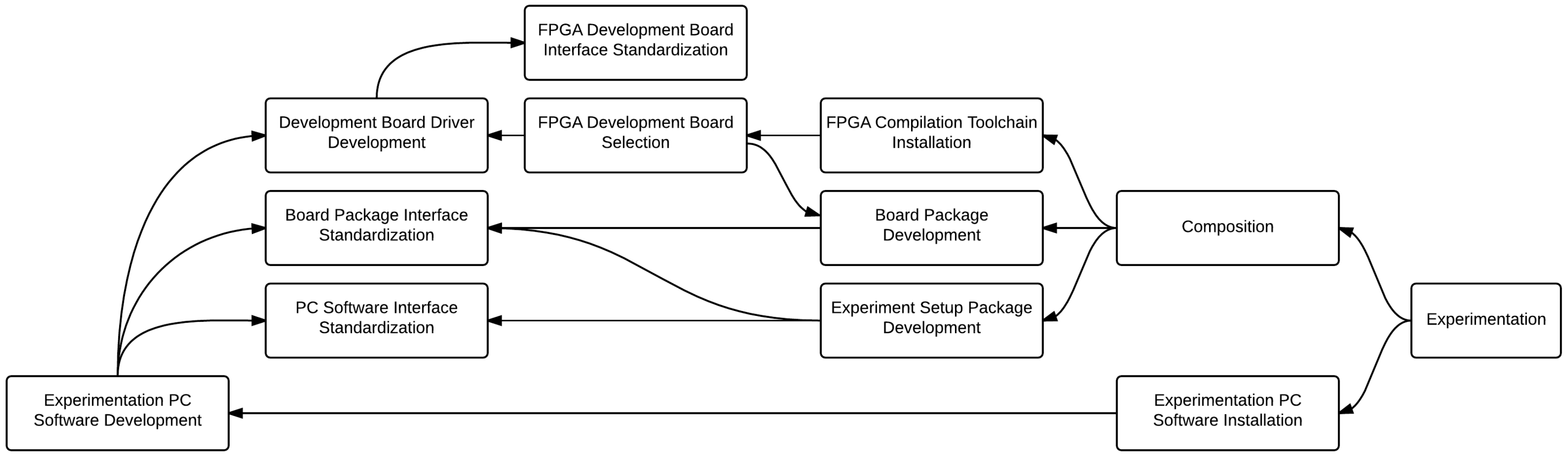

FPGA Development Board Interface Standardization
Development Board Driver Development
FPGA Development Board Selection
FPGA Compilation Toolchain Installation
Board Package Interface Standardization
Board Package Development
Composition
PC Software Interface Standardization
Experiment Setup Package Development
Experimentation
Experimentation PC Software Development
Experimentation PC Software Installation

Figure A.4: Controller abstracted, process dependency graph

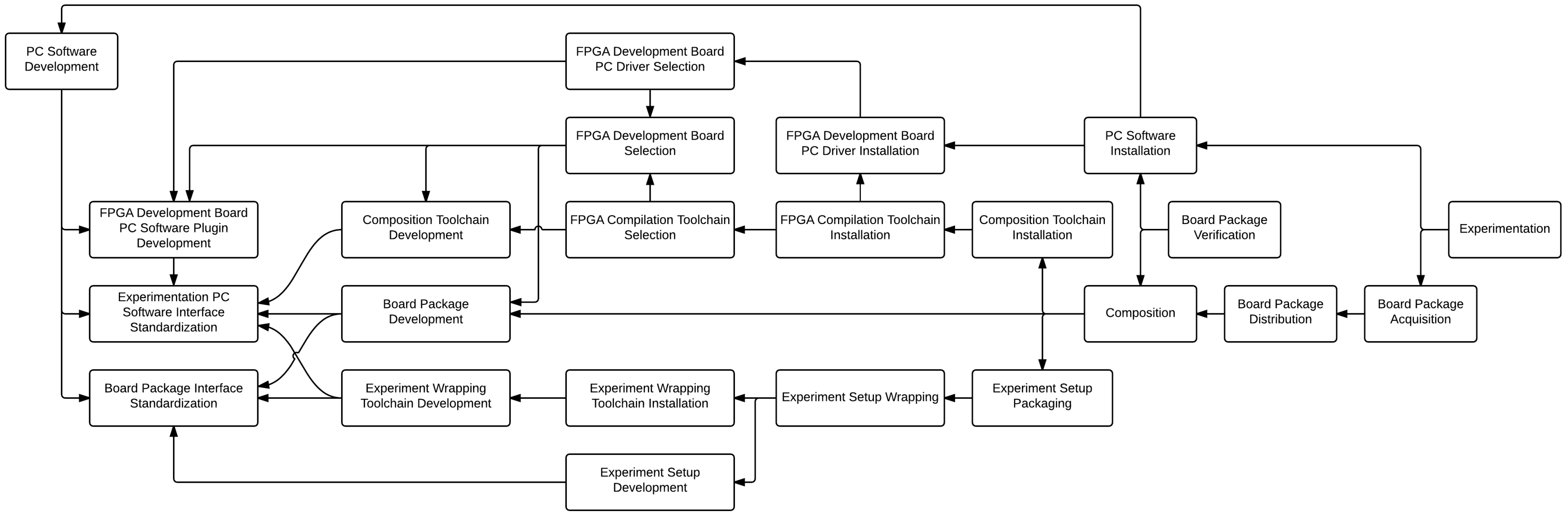

PC Software Development
FPGA Development Board PC Driver Selection
FPGA Development Board Selection
FPGA Development Board PC Driver Installation
PC Software Installation
FPGA Development Board PC Software Plugin Development
Composition Toolchain Development
FPGA Compilation Toolchain Selection
FPGA Compilation Toolchain Installation
Composition Toolchain Installation
Board Package Verification
Experimentation
Experimentation PC Software Interface Standardization
Board Package Development
Composition
Board Package Distribution
Board Package Acquisition
Board Package Interface Standardization
Experiment Wrapping Toolchain Development
Experiment Wrapping Toolchain Installation
Experiment Setup Wrapping
Experiment Setup Packaging
Experiment Setup Development

Figure A.5: I/O reintroduced, process dependency graph

APPENDIX B

Logic Diagrams

B.1 Simple wrapping

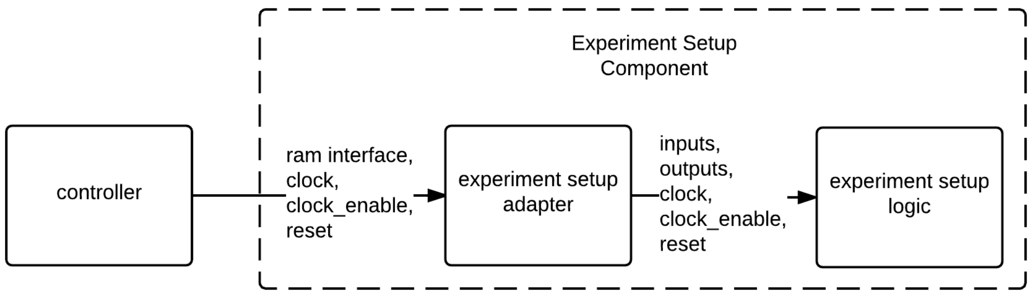


Figure B.1: Caption

B.2 Advanced wrapping

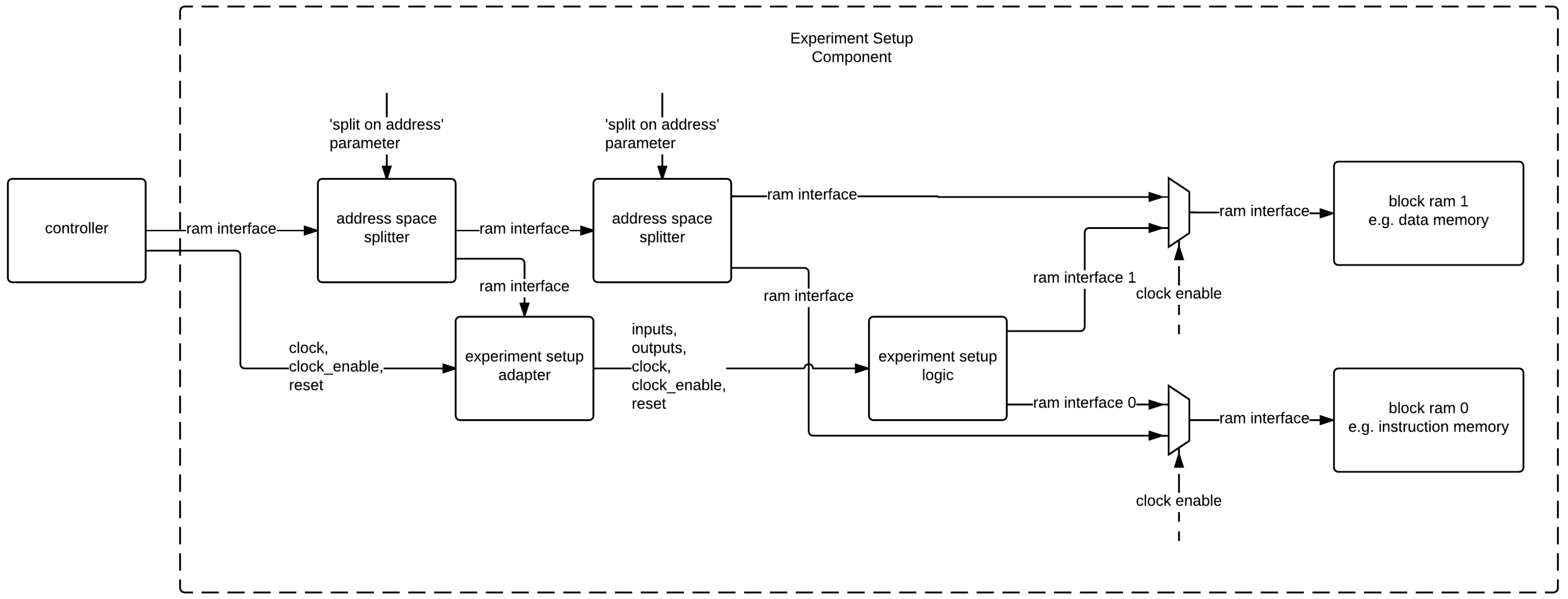


Figure B.2: Complex wrapper

B.3 Experiment Adapter

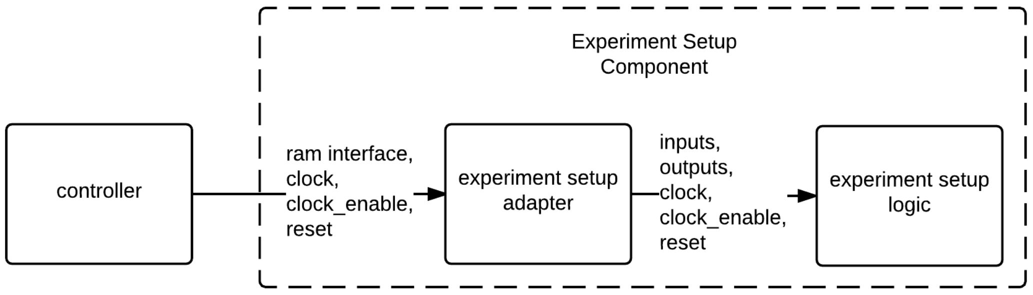

Experiment Setup Component
controller
ram interface, clock, clock_enable, reset
experiment setup adapter
inputs, outputs, clock, clock_enable, reset
experiment setup logic

Figure B.3: Caption

Bibliography

[1] J. Larus. (Jun. 2015). Spim mips simulator, [Online]. Available: `http://spimsimulator.sourceforge.net/`.

[2] (Jun. 2015). Sim-pl, [Online]. Available: `http://csa.science.uva.nl/sim-pl`.

[3] D. Jansen and B. Dusch, "Every student makes his own microprocessor," in *Microelectronics Education (EWME), 10th European Workshop on*, IEEE, 2014, pp. 97–101.

[4] M. D. L. A. Cifredo-Chacon, A. Quiros-Olozabal, and J. M. Guerrero-Rodriguez, "Computer architecture and fpgas: A learning-by-doing methodology for digital-native students," *Computer Applications in Engineering Education*, n/a–n/a, 2015, ISSN: 1099-0542. DOI: `10.1002/cae.21617`. [Online]. Available: `http://dx.doi.org/10.1002/cae.21617`.

[5] M. Pereira, P. Viera, A. Raabe, and C. Zeferino, "A basic processor for teaching digital circuits and systems design with fpga," in *Programmable Logic (SPL), 2012 VIII Southern Conference on*, Mar. 2012, pp. 1–6. DOI: `10.1109/SPL.2012.6211804`.

[6] A. El-Din and H. Krad, "Teaching computer architecture and organization using simulation and fpgas," *International Journal of Information and Education Technology*, vol. 1, no. 3, pp. 190–194, 2011.

[7] M. Holland, J. Harris, and S. Hauck, "Harnessing fpgas for computer architecture education," in *Microelectronic Systems Education, 2003. Proceedings. 2003 IEEE International Conference on*, IEEE, 2003, pp. 12–13.

[8] P. Bulić, V. Guštin, D. Šonc, and A. Štrancar, "An fpga-based integrated environment for computer architecture," *Computer Applications in Engineering Education*, vol. 21, no. 1, pp. 26–35, 2013.

[9] J. Xing, W. Zhao, and H. Hu, "An fpga-based experiment platform for multi-core system," in *Young Computer Scientists, 2008. ICYCS 2008. The 9th International Conference for*, IEEE, 2008, pp. 2567–2571.

[10] H. Oztekin, F. Temurtas, and A. Gulbag, "Bzk. sau. fpga10. 0: Microprocessor architecture design on reconfigurable hardware as an educational tool," in *Computers & Informatics (ISCI), 2011 IEEE Symposium on*, IEEE, 2011, pp. 385–389.

[11] C. M. Kellett, "A project-based learning approach to programmable logic design and computer architecture.," *IEEE transactions on education*, vol. 55, no. 3, pp. 378–383, 2012.

[12] J. H. Lee, S. E. Lee, H. C. Yu, and T. Suh, "Pipelined cpu design with fpga in teaching computer architecture," *Education, IEEE Transactions on*, vol. 55, no. 3, pp. 341–348, 2012.

[13] K. M. Al-Aubidy, "Teaching computer organization and architecture using simulation and fpga applications," *J. Comput. Sci*, vol. 3, no. 8, pp. 624–632, 2007.

[14] G. Wang, "Bridging the gap between textbook and real applications: A teaching methodology in digital electronics education," *Computer Applications in Engineering Education*, vol. 19, no. 2, pp. 268–279, 2011.

[15] K. Nakano and Y. Ito, "Processor, assembler, and compiler design education using an fpga," in *Parallel and Distributed Systems, 2008. ICPADS'08. 14th IEEE International Conference on*, IEEE, 2008, pp. 723–728.

[16] R. Paharsingh and J. Skobla, “A novel approach to teaching microprocessor design using fpga and hierarchical structure,” in *Microelectronic Systems Education, 2009. MSE’09. IEEE International Conference on*, IEEE, 2009, pp. 111–114.

[17] N. L. V. Calazans and F. G. Moraes, “Integrating the teaching of computer organization and architecture with digital hardware design early in undergraduate courses,” *Education, IEEE Transactions on*, vol. 44, no. 2, pp. 109–119, 2001.

[18] X. Inc. (Jun. 2015). What is an fpga? [Online]. Available: `http://www.xilinx.com/fpga/`.

[19] P. Kallstrom. (Jun. 2015). Fpga cell example, [Online]. Available: `https://commons.wikimedia.org/wiki/File:FPGA_cell_example.png`.

[20] M. Arora, *The Art of Hardware Architecture: Design Methods and Techniques for Digital Circuits*. Springer, 2011, ISBN: 1461403960.

[21] (). Rams, [Online]. Available: `http://www.csit-sun.pub.ro/courses/Masterat/Xilinx%20Synthesis%20Technology/toolbox.xilinx.com/docsan/xilinx4/data/docs/xst/hdlcode14.html` (visited on 02/22/2016).