Univerza *v Ljubljani*
Fakulteta za *matematiko in fiziko*

Oddelek za fiziko

Seminar 2 - 2nd year Master

Digital IC design using FPGA technology

Author: Michel Adamič
Mentor: izr. prof. dr. Andrej Trost[1]

Ljubljana, March 2019

Abstract

This seminar is a very basic introduction to digital integrated circuit (IC) design with Field Programmable Gate Arrays (FPGAs). It is intended for beginners without prior knowledge of the topic. The seminar first introduces digital electronics and integrated circuits in general. Next, FPGA devices are presented and how we can use them to implement simple digital circuits.

[1]Faculty of Electrical Engineering, University of Ljubljana

Contents

1 Introduction

Field Programmable Gate Arrays or FPGAs are modern devices which enable fast, efficient and relatively cheap implementation of digital integrated circuits. Traditionally used for prototyping, today's FPGAs are becoming more powerful than ever before, owing to the still ongoing Moore's law. Consequently, FPGA devices are finding their way into end products as well, offering a much more affordable solution compared to ASICs (Application Specific Integrated Circuits). This especially holds true for research institutions, where small batch sizes and budget limitations make FPGAs the only reasonable option for designing "in-house" digital integrated circuits.

The aim of this seminar is to familiarize the reader with this increasingly important technology, as there is a great chance one will come across it in the future. Starting with the basics of digital electronics and integrated circuits, we will proceed to FPGAs and how we can use them to implement simple digital circuits. We will learn that advanced Electronic design automation software tools are essential to master the ever increasingly complex hardware.

2 Digital electronics

2.1 Digital vs Analog

The world we live in is analog (neglecting quantum effects), meaning that various physical quantities we observe change continuously with time and can take any arbitrary value. In the world of electronics, the quantities forming electrical signals are voltage and current. Using analog circuits, we can amplify these signals, filter, add them, and even do basic differentiation or integration. Typical analog circuits are for example audio amplifiers, RF oscillators etc. However, possibilities end rather quickly. In addition, analog electronics are prone to noise, temperature and component variations etc.

The solution to these limitations are digital electronics, where a signal can only take discrete values. Today, we use the binary system, where a digital signal only exists in two possible states - a zero "0" or a one "1". For example, if a voltage is below a certain threshold, we declare it a "0", and if it's above

some other threshold, we declare it a ”1”. These threshold definitions are called logic levels. Figure 1 shows a classic example. Notice how the noise in the actual voltage doesn't affect the system at all.

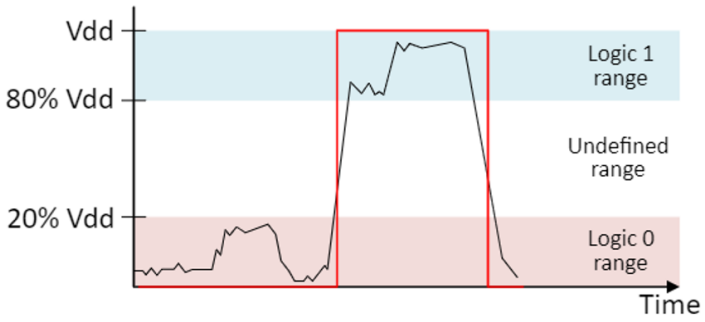


Figure 1: An example of a digitally interpreted signal according to the defined logic levels. The red line is the interpretation of the actual voltage, shown in black. [1]

2.2 The binary system

A digital value, represented with a ”0” or a ”1”, is called a *bit*. Stacking these bits together creates *words*. Words in real digital systems usually have standard lengths, like *bytes*, which are 8-bit words. But these things are not without a meaning; they are in fact mathematical numbers, expressed in the binary numeral system. For example, ”01001110” is equal to 78 in decimal or 4E in hexadecimal. The conversion from binary to decimal is straightforward, as shown below. [2]

$$\mathbf{01001110}\ bin = \mathbf{0}\cdot 1 + \mathbf{1}\cdot 2 + \mathbf{1}\cdot 4 + \mathbf{1}\cdot 8 + \mathbf{0}\cdot 16 + \mathbf{0}\cdot 32 + \mathbf{1}\cdot 64 + \mathbf{0}\cdot 128 = \mathbf{78}\ dec$$

All that needs to be done is to multiply bit values from right to left with increasing power of 2, as shown in the above example. The rightmost bit, or bit 0, is called the Least Significant Bit (LSB) and the leftmost the Most Significant Bit (MSB).

Digital is all about numbers, and having numbers means we can do math, using the so-called Boolean algebra [2]. For this, we make use of *logic gates*, digital circuits that perform basic Boolean or logic operations on binary values. In addition, bits can easily be stored on memory. The fact that we can represent and store numbers in binary and do math on them easily explains why digital circuits form the essence of computers.

What has just been told is the main advantage of digital circuits compared to traditional analog. Because digital electronics are based on mathematics, it means we can calculate or do or process anything we want, given enough time and/or resources. In fact, there are even functions that are impossible to implement in analog. This, together with insensitivity to noise and temperature variations, is the reason why the majority of electronics today are digital.

2.3 Combinational logic

As already mentioned, logic gates take binary values as inputs and perform logic operations upon them, producing binary outputs. Figure 2 shows some elementary gates and their corresponding truth tables.

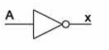
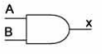
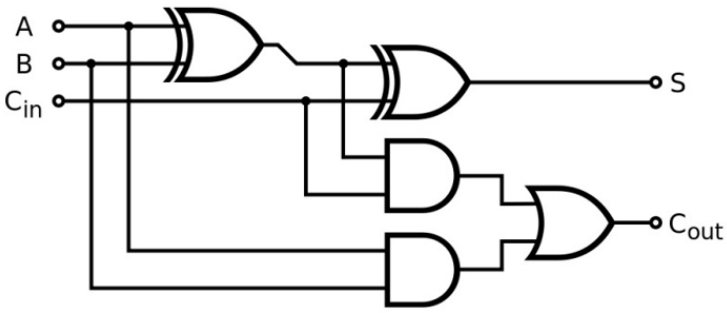
Name	NOT	AND	NAND	OR	NOR	XOR	XNOR
Alg. Expr.	$\overline{A}$	AB	$\overline{AB}$	$A+B$	$\overline{A+B}$	$A\oplus B$	$\overline{A\oplus B}$
Symbol							
Truth Table	A X 0 1 1 0	B A X 0 0 0 0 1 0 1 0 0 1 1 1	B A X 0 0 1 0 1 1 1 0 1 1 1 0	B A X 0 0 0 0 1 1 1 0 1 1 1 1	B A X 0 0 1 0 1 0 1 0 0 1 1 0	B A X 0 0 0 0 1 1 1 0 1 1 1 0	B A X 0 0 1 0 1 0 1 0 0 1 1 1

Figure 2: 2-input logic gates and their truth tables [3].

By stacking multiple logic gates together, any arbitrary Boolean function can be implemented. The output of such a circuit is exclusively dependent on the state of its inputs, as described by its truth table. This is called *combinational* logic [2]. As an example, Figure 3 shows a Full Adder circuit, which performs summation of two bits. It also includes carry lines to be connected to the neighbouring adders. Binary addition is one of the most basic mathematical operations and a fundamental part of every Arithmetic Logic Unit (ALU).



Inputs			Outputs	
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Figure 3: The Full Adder circuit [4] and its truth table [5].

2.4 Sequential logic

The output of most digital circuits doesn't depend solely on their inputs, but also on the *state* of the circuit, which is stored in some sort of memory. Such circuits belong to the class of *sequential* logic circuits [6]. A typical example of sequential logic is the pipeline structure, shown in Figure 4.

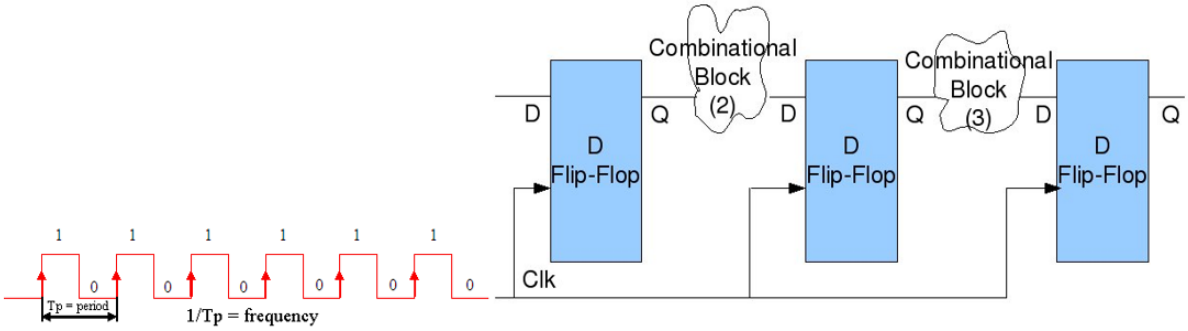


Figure 4: A pipeline structure, very commonly found in digital systems [7]. Shown on the left is the clock signal [8].

The basic building block of sequential circuits is the D flip-flop, the principal storage element in the digital world. Upon rising edge of the *clock* signal, the value on its D input is passed on to its Q output and stored there until the next rising edge of the clock. The clock is just a square wave with a certain period. By looking at Figure 4, we see that data gradually moves onto the next processing blocks of combinational logic with the steady pace of the clock signal. Therefore, the frequency of the clock dictates the speed of the circuit. This is called synchronous sequential logic, since the circuit operates synchronously with the clock. It turns out that most modern digital circuits are in fact synchronous, because if data values are discrete, it follows naturally to make time discrete as well.

3 Integrated circuits

3.1 An era of ICs

Since their invention and further developments in the 1960s [9], integrated circuits or ICs are the cornerstone of the modern technological revolution. Also called a chip or a microchip, an integrated circuit is a fully functional electronic circuit, integrated onto a single piece of semiconductor material, usually

silicon, with a typical size no larger than a fingernail. Figure 5 shows a micrograph of a chip from 1992. We notice a lot of tiny metal wires, interconnected orthogonally.

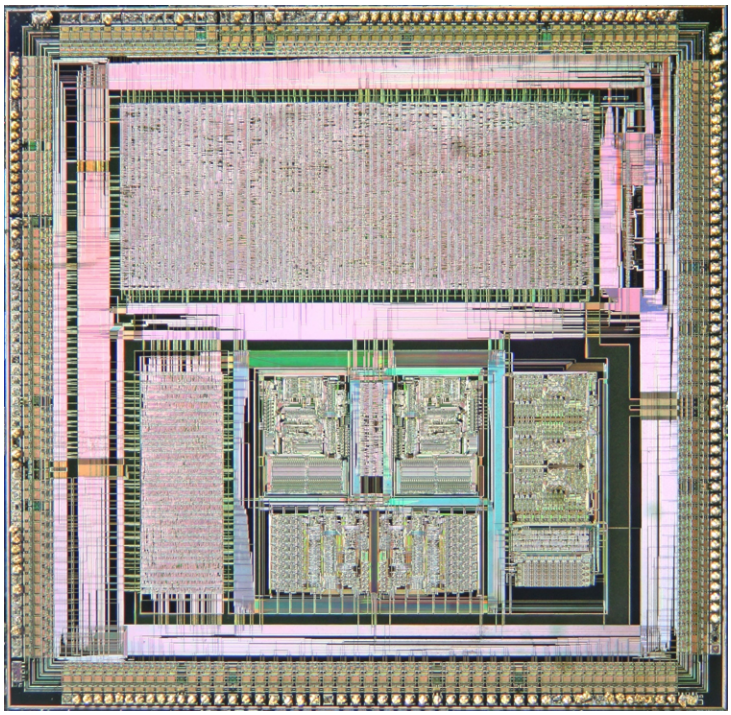


Figure 5: VLSI VL82C486, 1992, 1.0μm CMOS [10, 11].

ICs contain thousands, millions or even billions of tiny components, called transistors. These transistors in digital chips scale very well, applying Moore's law, which states that every two years, the number of components on a single chip doubles [12]. The Law has held for more than 50 years, and this continuous exponential growth has now reached a staggering number of 10 billion transistors on today's newest chips.

3.2 CMOS technology

The atom of modern integrated circuits is the CMOS inverter, a NOT gate, made out of two MOSFET transistors. A MOSFET, or metal-oxide-semiconductor field-effect transistor, is basically a silicon switch, which conducts between its terminals when a voltage is applied on its input, called the gate. The structure of the inverter is shown in Figure 6.

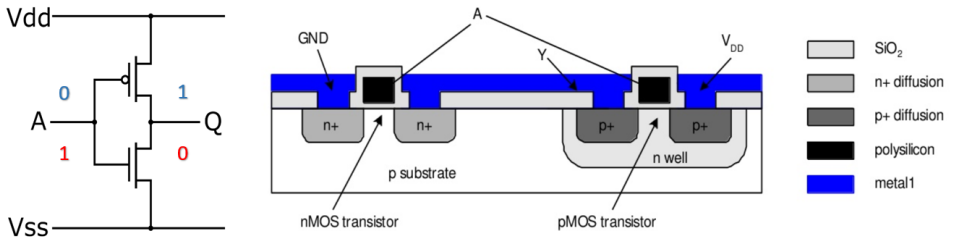


Figure 6: The CMOS inverter. Left: schematic [13], right: actual layout on the silicon substrate [14].

The name "CMOS" stems from the fact that two *complementary MOS* transistors are used: one is p-type (holes conduct) and the other is n-type (electrons conduct). This idea is used in other logic circuits as well, which include many such complementary pairs. The majority of today's digital integrated circuits are fabricated in CMOS technology, whose excellent characteristics have enabled the rapid advancement of ICs in the previous decades.

3.3 IC fabrication

Manufacturing modern ICs requires the most advanced science and engineering on the planet. The process begins with a silicon wafer, a thin disk of pure, monocrystalline silicon. These wafers then undergo hundreds of photolithographic steps in large, fully automated cleanrooms, called semiconductor foundries or "fabs". During the process, transistors are integrated into the silicon, followed by multiple layers of metal interconnect. Finally, after approximately a month, the wafer is ready to be diced into individual chips, which are then placed in a package. [15]

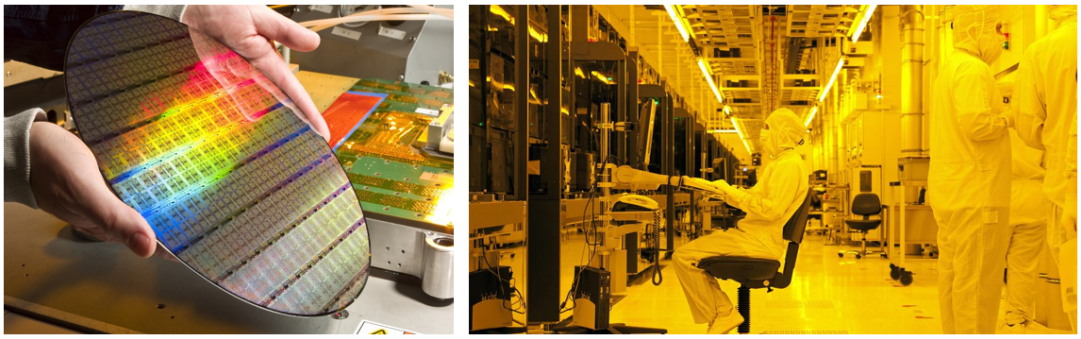


Figure 7: Left: a 300 mm Si wafer with multiple integrated die [16]. Right: GlobalFoundries Fab1 in Dresden, Germany [17].

4 Field Programmable Gate Arrays

4.1 Types of digital ICs

Before proceeding to FPGAs, we first need to understand a broader picture of various types of digital integrated circuits that exist today. Dividing digital circuits in different groups is tricky, since there are a lot of overlaps and possible ways of doing it. However, for the purpose of this seminar, we can roughly divide digital ICs in three groups [18, 19]:

- **Standard** circuits are universal, widely used digital chips that we can use to build bigger systems. They are easily accessible and cheap. Examples are logic gates, microprocessors, memory ...
- **Application-Specific Integrated Circuits** or **ASIC**s [11] are specialized circuits, made uniquely for a specific task. They offer highest performance, but have to be custom designed and are very expensive to produce. Examples include anything that isn't standard, from interface controllers to SoCs (System-on-Chips). The VLSI circuit from Figure 5 is an ASIC.
- **Programmable** circuits are digital ICs which are pre-made and only need to be programmed in order to implement a certain functionality. Consequently, circuit development is much faster and cheaper. There exist many classes of programmable circuits, but we will only mention **FPGA**s, since they offer the highest performance and are the most widely used.

For the rest of this seminar, we will focus on FPGAs and how to use them.

4.2 What is an FPGA?

A Field Programmable Gate Array is a fully programmable digital chip with a very regular structure, shown in Figure 8. It consists of Configurable logic blocks (CLBs), programmable interconnect and programmable input/output (I/O) blocks. When powered up, the FPGA won't do anything until programmed, or as we often say, configured properly. By default, FPGAs are fabricated in CMOS, meaning they are volatile devices, so the configuration has to be loaded every time the chip is powered on. The main players in the world of FPGAs are Xilinx and Intel (formerly Altera), both based in California.

The name "field-programmable gate array" originates from the fact that these devices can be programmed anywhere, any time, in other words "out in the field" and not just once in the factory [20]. The expression "gate array" refers to this matrix-like structure, found in FPGAs.

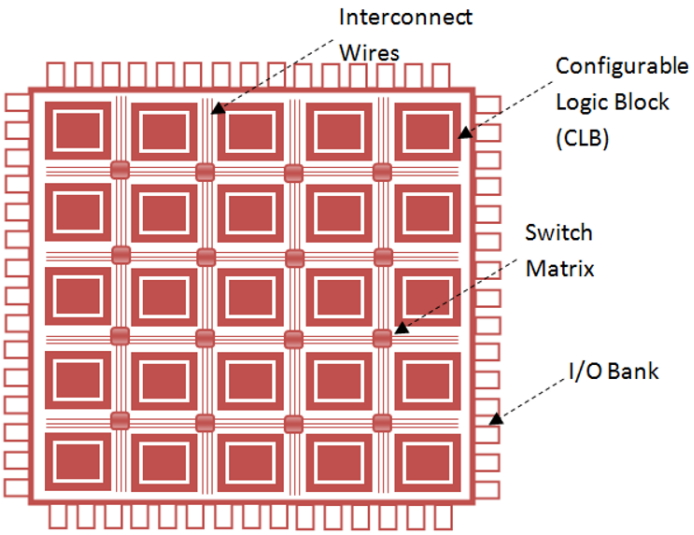


Figure 8: General FPGA architecture [21].

The heart of FPGA functionality are the Configurable logic blocks, which contain logic cells; their generic structure is shown in Figure 9. Inside, we find several distinct components. Lookup tables or LUTs are used to implement combinational logic. When configured, the LUTs basically serve as truth tables for a particular function. Next, there is a carry chain with the full adder (FA) circuit for fast arithmetic operations. Flip-flops are clocked storage elements for sequential logic and multiplexers (MUXes) route signals within the cell.

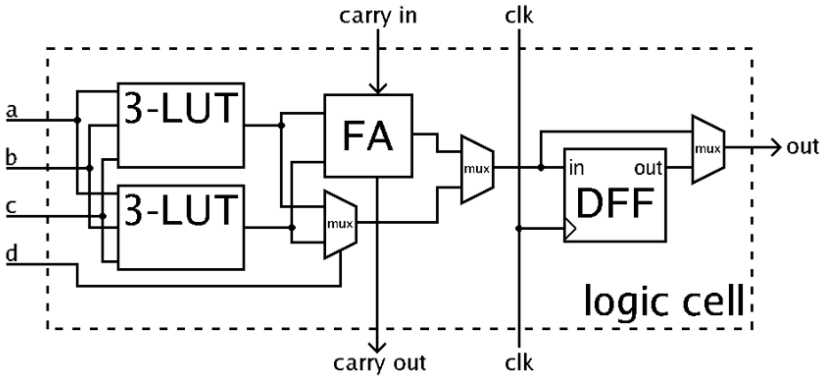


Figure 9: Generalized schematic of a logic cell - the basic unit of a CLB [22].

The cool thing is that we can implement *any digital circuit we want* on an FPGA, as long as the device has enough logic resources. FPGAs were traditionally used for prototyping ASIC designs, but due to marvellous developments in the semiconductor industry, they have now become powerful enough to form end products as well [11]. Compared to ASICs, designing digital systems with FPGAs is much easier, cheaper and faster, making them a very popular choice in many areas. Moreover, state-of-the-art FPGAs are even starting to rival ASICs in performance. On the downside, FPGAs are usually slower than ASICs, use more silicon, and have a larger power consumption [18]. However, modern FPGAs are extremely powerful devices, which also contain Block Random Access Memory (BRAM), Digital Signal Processing (DSP) blocks and often even hard Central Processing Unit (CPU) cores. In the following chapter, we will take a look at how we can design our own digital circuits using an FPGA.

5 FPGA design flow

Today's integrated circuits have become so incredibly complicated that we have to rely on Electronic design automation (EDA) to be able to design an IC [23]. In the case of FPGAs, the circuit to be imple-

mented is designed using sophisticated software tools, which are provided by the device manufacturer. We will now go through crucial steps [24], leading from circuit description to a fully programmed FPGA, using examples from Xilinx *Vivado*.

5.1 Design entry

When we have decided and specified what circuit we want to make, the design process can begin. First, we somehow have to "draw" the electrical schematic. People actually used to do that, but today, as ICs got more complex, that is not common practice any more. [20] Instead, we use a Hardware Description Language, or HDL, for example VHDL or Verilog. A HDL somewhat looks like a programming language, but it is not. With HDL, we describe the functionality of our digital circuit, which will be synthesized later on from this description. For example, Figure 10 shows two very primitive designs in VHDL - one is an AND gate and the other a D flip flop.

```vhdl
-- Just a lonely AND gate

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity AndGate is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           y : out STD_LOGIC);
end AndGate;

architecture opis of AndGate is
begin
   y <= a and b;
end opis;
```

```vhdl
-- Just a lonely D flif-flop

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity DFF is
    Port ( d : in STD_LOGIC;
           clk : in STD_LOGIC;
           q : out STD_LOGIC);
end DFF;

architecture opis of DFF is
begin
   process(clk)
   begin
      if rising_edge(clk) then
         q <= d;
      end if;
   end process;
end opis;
```

Figure 10: VHDL example code for an AND gate (left) and a D flip flop (right).

Each VHDL file consists of two important declarations: *entity* and *architecture*. Within entity, we give our circuit a name and declare its input/output pins. In the case of the AND gate, we have two inputs, A, B, and a Y output. As for the DFF, D and CLK are the inputs, while Q is the output. Next, we specify how the circuit works within the architecture environment. In this case, both architectures are fairly trivial.

Describing digital circuits on the level of gates and flip flops (also called registers), as we just saw, is known as the Register-transfer level (RTL) abstraction of digital IC design [25]. It is the level of abstraction on which hardware description languages operate. What Figure 10 shows are very primitive examples. Of course, with real systems, we use higher level HDL features, which enable more efficient design on more algorithmic levels. But since this seminar is not about HDL, we will stop here.

Once we have finished the design of our digital circuit using HDL, the source code will, provided there are no syntax errors, "compile" itself to a RTL netlist, as shown in Figure 11. In other words, the EDA tool will read the HDL code and turn it into an RTL schematic of the circuit we described. For a human, this is much easier than drawing the schematic from scratch. Instead, we circuit designers rather describe the functionality of the circuit on a higher level and let the machine draw the diagram. At this point it needs to be emphasized that the register transfer level netlist is a generic schematic which uses generic RTL components, like gates, multiplexers, registers, adders, multipliers etc. It has nothing to do with FPGAs. Yet.

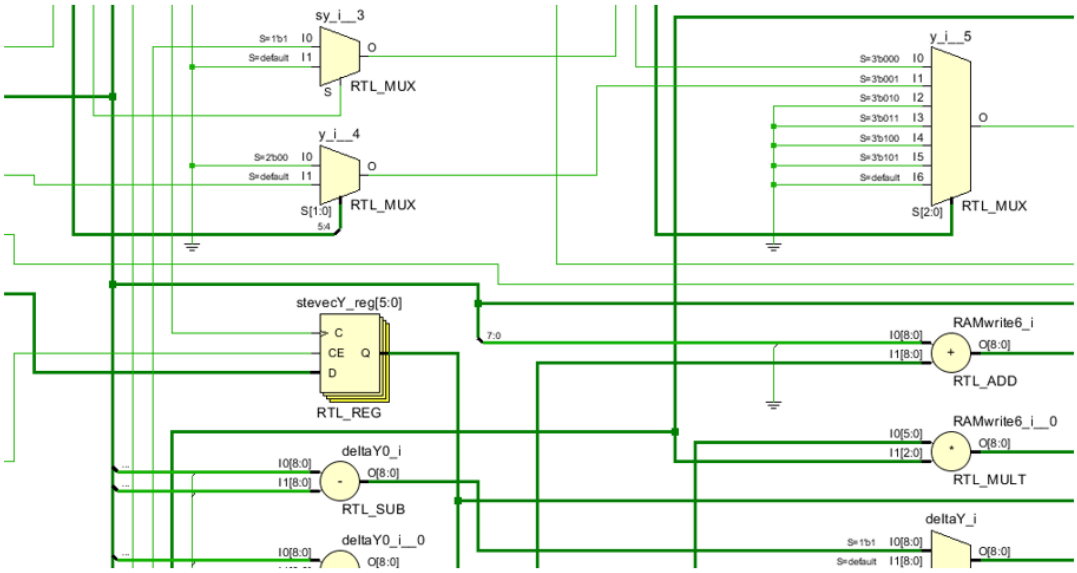


Figure 11: An example register transfer level schematic, as shown by Xilinx Vivado.

5.2 Behavioral simulation

In the next step, the circuit designer checks if the circuit functionally behaves as expected, using a simulation. This is done by writing a test bench, where we send various signals at the circuit inputs and check for the right outputs. If something shows wrong, we have to go back and modify the HDL design. An output of a typical simulation in Vivado is shown in Figure 12. It enables us to observe every signal in the circuit at any point in time.

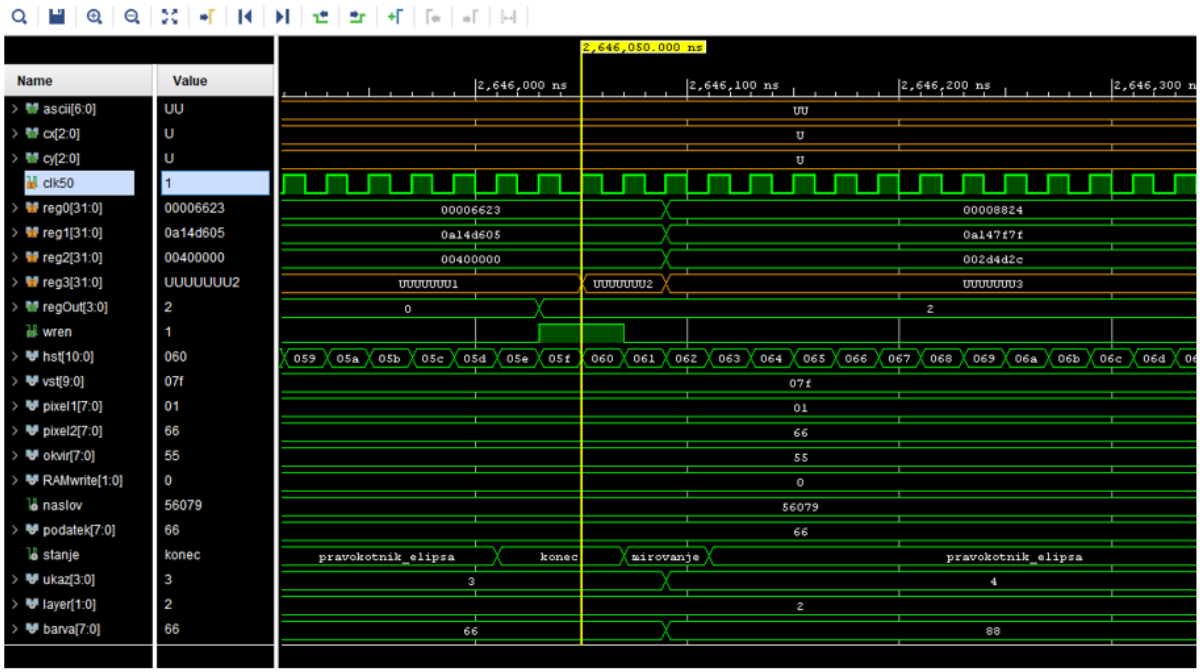


Figure 12: A behavioral simulation window in Xilinx Vivado.

It should be noted here that the behavioral simulation is only used to check if the designed circuit behaves as expected on the functional level, in order to check for ”bugs” in the HDL code. The simulation is purely theoretical and provides no guarantee at all that the circuit will actually work when implemented on the FPGA.

5.3 Logic synthesis

The next big step is the logic synthesis. Our job at this point is mostly done and it is now time to let the computer do its magic. During this step, the EDA tool will take the RTL schematic, created previously, and translate it to the target technology. In this case, the tool maps generic components used in RTL, like in Figure 11, to primitives available within the FPGA, namely lookup tables, multiplexers, registers etc.; the components available in CLBs from Figure 9. For example, combinational logic (RTL logic gates) will be translated to LUTs. For basic arithmetic functions, the carry chain will be used. More advanced mathematical operations, like multiplication, will probably occupy the DSP (digital signal processing) blocks, etc. Figure 13 shows a section of a synthesized netlist. As we can see, the RTL components have been replaced with specific components available in the target FPGA.

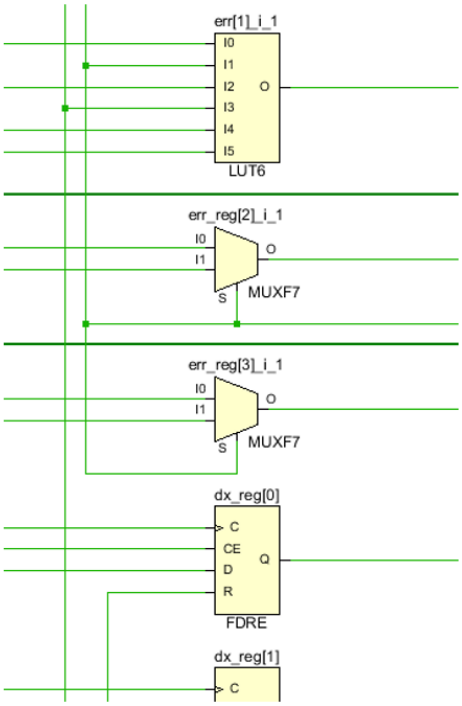


Figure 13: A small section of a synthesized netlist. We notice some registers, MUXes and a 6-input LUT.

5.4 Implementation and FPGA programming

We have now arrived at the final and at the same time most critical and computationally intensive step, called design implementation. During this step, also known as ”place & route”, the synthesized netlist is mapped onto the physical logic cells inside the FPGA. That is to say, every component from Figure 13 is assigned a specific physical location inside the chip and connected together. This is done according to the design *constraints* we have specified before the implementation. In the constraints file, the circuit designer specifies which outside pins on the FPGA the circuit should connect to, and most importantly, at what clock speed we would like the circuit to run on. For example, we have designed a circuit and would like it to run on the 100 MHz clock signal that is available from the board crystal. The EDA software tool will read these constraints and try to do its best to meet them. It will search through multiple possible layouts of the circuit until it finds one that is fast enough to be able to run on 100 MHz. It’s an optimization problem without a very well defined minimum. Usually, the tool just finds a local minimum, so the implemented circuit probably won’t be ”the best in the world”, but rather ”good enough” to meet the constraints. It is also possible to set the implementation tool for different optimizations, for example minimum power or area, maximum performance etc. The implementation step is very computationally intensive and can take hours, or even days for very large designs. This is also one of the reasons why doing simulations before actually implementing the design is important. Imagine making a minor change to the HDL code and then having to wait for ages to see the effect, only to discover the bug is still there ... That would be a proper nightmare.

When the implementation step is completed, we can see what the software tool did. Figure 14 shows an example of an implemented design, where the device layout is shown and which resources have been used to implement the circuit (highlighted in blue).

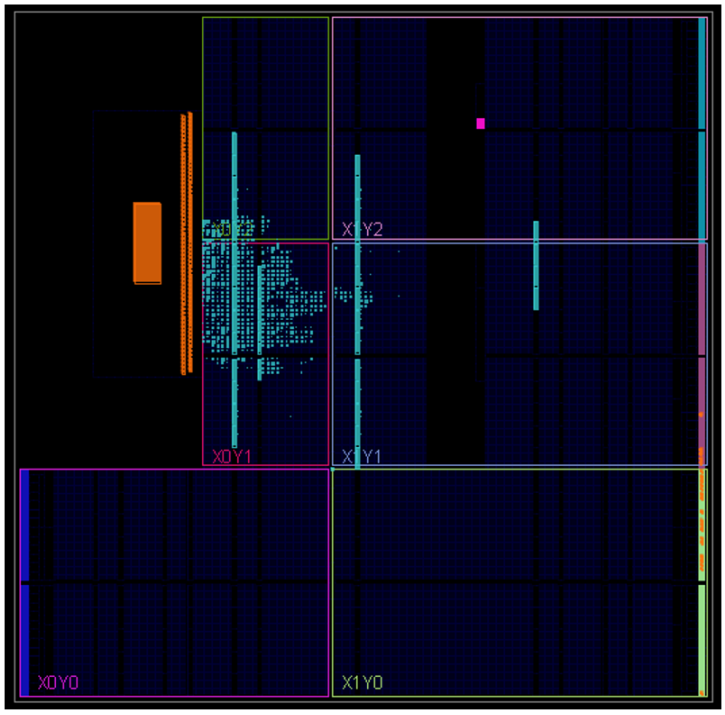


Figure 14: Implemented design on a Xilinx Zynq 7020 FPGA.

In addition, Vivado produces a lot of reports, where we can find detailed information about the implemented circuit. This includes:

- Resource utilization, which shows how many FPGA logic resources, like LUTs, flip flops, BRAM blocks etc. have been used.
- Timing information. Were the specified clock constraints met? Which signal path is the most critical (takes the longest time)? Here we make sure that the circuit will not be overclocked.
- Power estimation. The software calculates how much power dissipation we can expect from the chip, to make sure it won't overheat.
- ... and more ...

Once we are happy with the design, a "bitstream" file is created, which is a binary file containing the FPGA configuration. Then, it's time to program the device and see if our circuit actually works in practice! Ideally, we can use a development board like the one in Figure 15 for prototyping.

6 High-level synthesis (HLS)

As hardware advances, so do the design tools. Today, it is possible to implement a digital circuit from scratch using ordinary high level programming languages, like C++, instead of bothering with rather counterintuitive HDLs. This is called high level synthesis, or HLS, and it offers the design of digital ICs on a more algorithmic level, which is certainly more familiar to humans than RTL. HLS tools take the C (or whatever) code and synthesize the RTL level description in HDL from it automatically [26]. This makes the design of data processing circuits much easier, but the circuit designer loses control of the fine details of how the design will actually be implemented. For this reason, hardware description languages are still in widespread use. High level synthesis comes in handy for hardware acceleration of certain tasks on FPGAs, like neural networks, bitcoin mining and so on, to name a few.

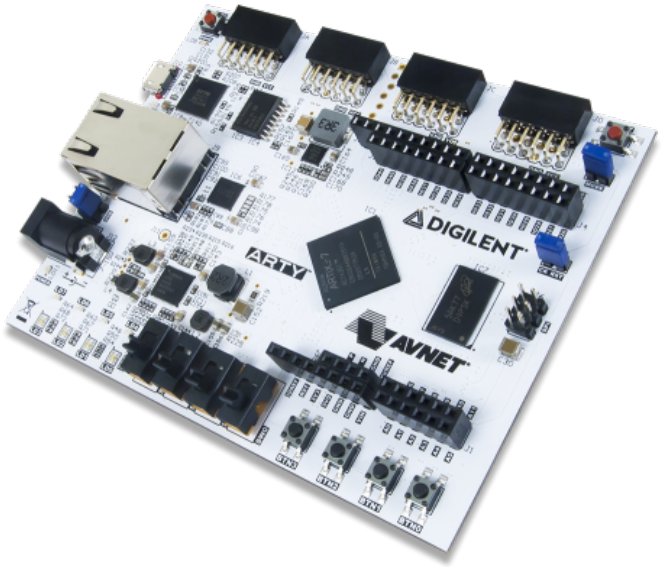


Figure 15: Development board with an Artix 7 family FPGA from Xilinx [27].

7 Conclusion

Designing digital ICs is becoming more and more abstract, thanks to the advancement of Electronic design automation tools. In combination with cheap and powerful programmable logic, like FPGAs, digital IC design has become more accessible than ever before. The bond with hardware is getting loose and today, you don't need to know how an FPGA works in order to use it. You just need to specify the behaviour of your digital circuit with a high level description, sit back, and let the EDA do the rest. Consequently, non-experts can get their simple digital designs up and running on an FPGA in a couple of minutes.

The main goal of this seminar was to break the ice between a beginner and the advanced technology of Field Programmable Gate Arrays. The reader should now know the basics of digital integrated circuits and is ready to start exploring their implementation on FPGA devices.

Happy designing!

References

[1] Electronic Signals. https://www.realdigital.org/doc/ee55dfd5145062f20379a773b14ad9b3. Accessed: 20.3.2019.

[2] Tadej Kotnik. Digitalne strukture. http://lbk.fe.uni-lj.si/pdfs/DS-Predavanja-2011-2012.pdf. Učno gradivo s predavanj, 2011/12.

[3] V. V. Peetham. How to teach logic to your #NeuralNetwork. https://medium.com/autonomous-agents/how-to-teach-logic-to-your-neuralnetworks-116215c71a49, Aug 14, 2016. Accessed: 20.3.2019.

[4] Adam Fabio. Create a full adder using the C preprocessor. https://hackaday.com/2013/10/09/create-a-full-adder-using-the-c-preprocessor/, Oct 9, 2013. Accessed: 20.3.2019.

[5] Adder (electronics). https://en.wikipedia.org/wiki/Adder_(electronics). Accessed: 20.3.2019.

[6] Sequential logic. https://en.wikipedia.org/wiki/Sequential_logic. Accessed: 20.3.2019.

[7] Ujjwal Chand. Pipelining and Retiming in High Speed Digital Logic Designs. http://chandujjwal.blogspot.com/2009/02/pipelining-and-retiming-in-high-speed.html, Feb 1, 2009. Accessed: 20.3.2019.

[8] https://archive.cnx.org/contents/bebf3577-02ca-489e-9197-67940ab171f8@1/3-1-what-is-a-digital-clock. Accessed: 20.3.2019.

[9] Integrated circuit. https://en.wikipedia.org/wiki/Integrated_circuit. Accessed: 20.3.2019.

[10] VLSI Technology Inc. Advance information - VL82C486, June 1992. Product datasheet.

[11] Application-specific integrated circuit. https://en.wikipedia.org/wiki/Application-specific_integrated_circuit. Accessed: 20.3.2019.

[12] Moore's law. https://en.wikipedia.org/wiki/Moore%27s_law. Accessed: 20.3.2019.

[13] CMOS. https://en.wikipedia.org/wiki/CMOS. Accessed: 20.3.2019.

[14] Jihyasha Maru. Fabrication of CMOS. https://www.slideshare.net/jigyashamaru/cmos-fabrication. Slides, accessed: 20.3.2019.

[15] GlobalFoundries. Sand to Silicon. https://www.youtube.com/watch?v=UvluuAIiA50, May 9, 2012. Accessed: 20.3.2019.

[16] Q4 2017 300 mm Silicon Wafer Pricing to Increase 20% YoY in DRAM-like Squeeze. TechPowerUp, https://www.techpowerup.com/238785/q4-2017-300-mm-silicon-wafer-pricing-to-increase-20-yoy-in-dram-like-squeeze, Nov 14, 2017. Accessed: 20.3.2019.

[17] GlobalFoundries spurns offer to take on IBM factories. The National, https://www.thenational.ae/business/globalfoundries-spurns-offer-to-take-on-ibm-factories-1.324268?videoId=5605357113001, Aug 5, 2014. Accessed: 20.3.2019.

[18] Andrej Trost. Programirljiva vezja. https://lniv.fe.uni-lj.si/courses/des/DES17_FPGA.pdf. Digitalni elektronski sistemi, 2017.

[19] LNIV FE. https://lniv.fe.uni-lj.si/courses/iv/Zasnova.pdf. Digitalna integrirana vezja in sistemi, accessed: 20.3.2019.

[20] Field-programmable gate array. https://en.wikipedia.org/wiki/Field-programmable_gate_array. Accessed: 20.3.2019.

[21] Shahul Akthar. FPGA Architecture. https://allaboutfpga.com/fpga-architecture/, April 16, 2014. Accessed: 20.3.2019.

[22] Logic block. https://en.wikipedia.org/wiki/Logic_block. Accessed: 20.3.2019.

[23] Electronic design automation. https://en.wikipedia.org/wiki/Electronic_design_automation. Accessed: 20.3.2019.

[24] Xilinx. FPGA Design Flow Overview. https://www.xilinx.com/support/documentation/sw_manuals/xilinx10/isehelp/ise_c_fpga_design_flow_overview.htm, 2008. Accessed: 20.3.2019.

[25] Register-transfer level. https://en.wikipedia.org/wiki/Register-transfer_level. Accessed: 20.3.2019.

[26] High-level synthesis. https://en.wikipedia.org/wiki/High-level_synthesis. Accessed: 20.3.2019.

[27] Digilent. Arty Programming Guide. https://reference.digilentinc.com/learn/programmable-logic/tutorials/arty-programming-guide/start. Accessed: 28.3.2019.