FPGA and FPGAC(High level synthesise tools)

Abstract

FPGA stands for Field Programmable Gate Array. It is an integrated circuit that can be configured by the user in order to implement digital logic functions of varying complexities. This report concerns FPGAs (Field Programmable Gate Arrays). The basic FPGA blocks, I/O, CLBs (Combinational Logic Blocks), and architecture, the advantages of FPGAs are discussed to impart a basic understanding of FPGA operation. Digital signal processing is an important area where FPGAs have found many applications in recent years. Interest in high-level synthesis tools for FPGAs is intensifying as FPGAs and their applications grow larger and more complex. Prospective users want to understand how well high-level synthesis tools work, both in terms of usability and quality of results. To meet this need, I have prepared this paper that presents a brief introduction to the FPGA.

1 Introduction

FPGAs are reprogrammable silicon chips. Using prebuilt logic blocks and programmable routing resources, you can configure these chips to implement custom hardware functionality without ever having to pick up a breadboard or soldering iron. You develop digital computing tasks in software and compile them down to a configuration file or bitstream that contains information on how the components should be wired together. In addition, FPGAs are completely reconfigurable and instantly take on a brand new "personality" when you recompile a different configuration of circuitry.

In the past, FPGA technology could be used only by engineers with a deep understanding of digital hardware design. The rise of high-level design tools, however, is changing the rules of FPGA programming, with new technologies that convert graphical block diagrams or even C code into digital hardware circuitry. FPGA chip adoption across all industries is driven by the fact that FPGAs combine the best parts of ASICs and processor-based systems.

FPGAs provide hardware-timed speed and reliability, but they do not require high volumes to justify the large upfront expense of custom ASIC design. Reprogrammable silicon also has the same flexibility of software running on a processor-based system, but it is not limited by the number of processing cores available. Unlike processors, FPGAs are truly parallel in nature, so different processing operations do not have to compete for the same resources. Each independent processing task is assigned to a dedicated section of the chip, and can function autonomously without any influence from other logic blocks. As a result, the performance of one part of the application is not affected when you add more processing.[1]

2 Features of FPGA

A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by a customer or a designer after manufacturing.

FPGAs have large resources of logic gates and RAM blocks to implement complex digital computations.

FPGAs contain programmable logic components called "logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together". Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory Due to their nature, processors execute commands in a sequential manner; you can places as many commands as you wish on a storage, and ask your processor to read the storage and execute them one-by-one. This leads into a productive task being done, such as booting a computer, or showing your operating system on your screen.

An FPGA is a silicon wafer, which can have up to several hundred thousand digital Cells. Each cell can perform simple tasks such as adding 2 bits, keeping 8 bits (acting as a very small RAM) or acting as a small multiplexer, decoder or several gates. However, initially they are not connected to each other. You can design a program that commands each cell what task to perform, and also connect the to other cells in the correct order. FPGAs are programmable, and their interconnection will shaped based on your design. They must be programmed each time on startup, since most of them cannot retain their configuration after power-down, as they use RAM technology.

FPGA can have up to several hundred thousand digital cells. This gives you the ability to create 100 multipliers, that each can show the multiplication of two numbers at the same time. This is the very nature of FPGAs; the ability to perform unlimited tasks in parallel, something that cannot be achieved in processors, as they can execute instructions one-by-one in a sequential manner.[1] [2]

The FPGA configuration is generally specified using a hardware description language (HDL).

What is hardware description language (HDL)?

In electronics, a hardware description language (HDL) is a specialized computer language used to program the structure, design and operation of electronic circuits, and most commonly, digital logic circuits.

A hardware description language enables a precise, formal description of an electronic circuit that allows for the automated analysis, simulation, and simulated testing of an electronic circuit. It also allows for the compilation of an

HDL program into a lower level specification of physical electronic components, such as the set of masks used to create an integrated circuit.

A hardware description language looks much like a programming language such as C; it is a textual description consisting of expressions, statements and control structures. One important difference between most programming languages and HDLs is that HDLs explicitly include the notion of time.

HDLs form an integral part of Electronic design automation systems, especially for complex circuits, such as microprocessors.[3]

2.1 Who makes FPGA in market ?

FPGA vendors:

1. Xilinx
2. Altera
3. QuickLogic
4. Achronics (Specialized in very high speed FPGA)
5. Lattice

• Xilinx and Altera are the main manufacturers.

• You can find their ICs on EBay, Arrow and other major electronic distributors.

3 Technologies similar to FPGA

There are two technologies which can be used for the same purposes instead of FPGA.

(1) Application Specific Integrated Circuits (ASIC)

(2) General purpose processors (GPP)

3.1 ASIC

• An application-specific integrated circuit (ASIC) is an integrated circuit (IC) customized for a particular use, rather than intended for general-purpose use. For example, a chip designed to run in a digital voice recorder is an ASIC.[4]

3.2 FPGA v/s ASIC

• Field Programmable Gate Arrays (FPGAs) and Application Specific Integrated Circuits (ASICs) provide different values to designers, and they must be carefully evaluated before choosing any one over the other.

• Deciding between ASICs and FPGAs requires designers to answer tough questions concerning costs, tool availability and effectiveness, as well as how best to present the information to management to guarantee support throughout the design process.

• The FPGA design flow eliminates the complex and time-consuming floorplanning, place and route, timing analysis, and mask / re-spin stages of the

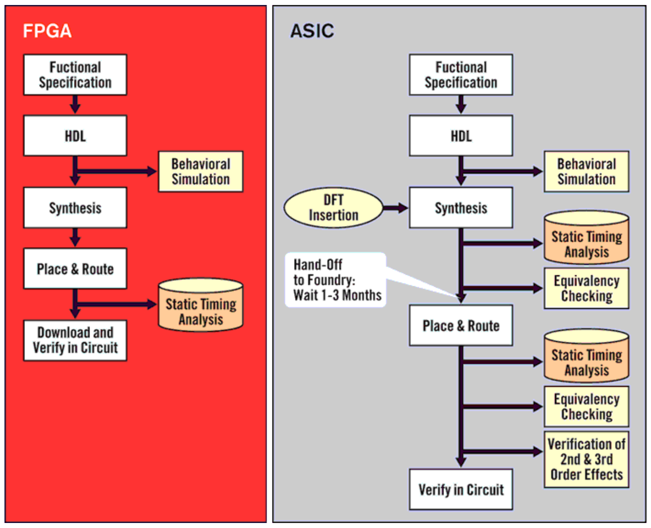


Figure 1: Asic v/s FPGA

project since the design logic is already synthesized to be placed onto an already verified, characterized FPGA device. However, when needed, Xilinx provides the advanced floorplanning, hierarchical design, and timing tools to allow users to maximize performance for the most demanding designs.[5]

3.3 GPP

General purpose processors (GPP) are designed for general purpose computers such as PCs or workstations.All techniques that can increase CPU speed have been applied to GPPs. For example, GPPs usually include on-chip cache and on-chip DMAs. Commonly used math operations are also supported by the on-chip hardware. GPPs are not designed for fast real-time applications.[7]

GPP is a general-purpose preprocessor with customizable syntax, suitable for a wide range of preprocessing tasks. Its independence from any one programming language makes it much more versatile than the C preprocessor (cpp), while its syntax is lighter and more flexible than that of GNU m4. There are built-in macros for use with C/C++, LaTeX, HTML, XHTML, and Prolog files.GPP is Free Software. It is distributed under the terms of the GNU General Public Licence.

General Purpose Preprocessor (GPP) is a macro processor that is not tied to or integrated with a particular language or piece of software. A macro processor is a program that copies a stream of text from one place to another, making a systematic set of replacements as it does so. Macro processors are often embedded in other programs, such as assemblers and compilers. Sometimes they are standalone programs that can be used to process any kind of text.

Macro processors have been used for language expansion (defining new language constructs that can be expressed in terms of existing language components), for systematic text replacements that require decision making, and for text reformatting (e.g. conditional extraction of material from an HTML file).[6] In addition to macros, GPP understands comments and strings, whose syntax and behavior can be widely customized to fit any particular purpose.

3.4 Difference between FPGA, ASIC and GPP

FPGA, ASIC and GPP are differentiated by following factors:

(1) Performance:

FPGA is excellent in the matter of performance. With FPGAs, developers can tune hardware gates specific to the application, delivering high application-specific performance.

ASIC is excellent in the matter of performance. For ASIC, like FPGA, you can tune hardware gates specific to your application offering high application-specific performance.

GPP is good in the matter of performance. GPP's typical high Megahertz levels will yield decent signal processing, but typical lack of mathematical specific single cycle instructions and multiplier units limit their real-time performance.

(2) Power efficiency :

FPGA is poor in the matter of power. FPGA is the bottom of the pack in power efficiency, an inherent trait in FPGA circuit technology as well as the overhead power of unused gates in the array. Technology advances will lower FPGA power, but likely not enough to change its place in the relative ranking on power efficiency.

ASIC is good in the matter of power. ASIC can achieve decent power efficiency when the design is specifically targeted for power efficiency, similar to the position on configurable. However, most ASIC-oriented designs are focused on performance or recurring cost (price) reasons, not power.

GPP is fair in the matter of power. Typically, GPP are general in nature, making them less power efficient than DSPs, ASSPs or what configurable should be able to achieve.

(3) Feature Flexibility:

FPGA is good in the matter of feature flexibility. It can be field reconfigured for additional features or changes.

ASIC is poor in the matter of feature flexibility. With ASIC, once you commit to your logic for your ASIC, you have to start over and redesign to add features.

GPP is excellent in the matter of feature flexibility. GPP are programmable processors and thus can utilize software programmability to achieve different functions and features, saving time versus similar hard coded logic implementations.

(4) Price:

FPGA is poor in the matter of price. FPGA is by far the most expensive alternative discussed here, achieving a poor rating.

ASIC is excellent in the matter of price, the only alternative to do so. In fact, dedicated standard cell logic gates with no overkill or underkill are the most efficient and smallest semiconductor chip size, thus likely offer the lowest recurring price. Also, ASIC commoditization can drive down the price per gate of ASIC chips.

GPP is fair in the matter of price. GPP processors have significant general purpose functions on board that tend to make them good for desktop applications, but only fair on price effectiveness for real-time signal processing.

(5) Time to Market:

FPGA is good in the matter of time to market. FPGAs avail field-ready modifications to achieve functions. Their flexibility is not as high as software programmable alternatives, thus they fall below GPP in their rating for time to market. However, they have better support and faster cycle time than ASSP, configurable processors or ASIC, and thus can claim faster time to market than those alternatives.

ASIC is poor in the matter of time to market. Configurable processors and ASICs are similar because the long cycle time required to complete and test an ASIC or configurable processor, fab the wafers unique to your design, and validate your solution can be many months to years. Whether configuring from gates or higher-level processor elements, they are only slightly different in degree of difficulty.

GPP is good in the matter of time to market. GPP processors are all programmable processors and thus can utilize software programmability to achieve different functions and features, saving time to market versus similar hard coded logic implementations.

(6) Development ease: FPGA is excellent in the matter of development ease. FPGA would rate the best on development cost assuming two situations: that the toolset for FPGA programming is not too expensive; and, assuming the developer is dealing primarily with hardware, that the engineer is involved in the development. If development leans towards software engineers, then FPGA would increase in effort and relative cost. In terms of development help, the FPGA tools and support structure for FPGA-based designs seems to be well established and acceptable to OEMs.

ASIC is fair in the matter of development ease. Although ASIC can be perceived as inexpensive, it is in fact the most expensive when it comes to overall development cost. This is attributed to the amount of logic design that has to be done by the customer to create the application on the chip, coupled with the ever increasing cost of silicon processing, with multiple hundreds of thousands of dollars per full reticle revision of silicon. In terms of development help, ASIC provides general support, but does not offer any application-specific help due to general lack of application-specific content knowledge at ASIC suppliers.

GPP is good in the matter of development ease. GPP programmability of existing chips allows for faster development cycles for the desired function versus typically developing application-specific chips or ASICS. With proper use of high-level programming and/or use of standard code modules, one can cut development time significantly, and thus save development cost. In terms of development help, GPP designers receive general support, but don't get any application-specific or real-time help due to general lack of application-specific and real-time content knowledge at GPP suppliers.

4 Advantages of FPGA Technology

There are 5 advantages of FPGA which are as following.

1. Performance
2. Time to Market
3. Cost
4. Reliability
5. Long-Term Maintenance

4.1 Performance

Taking advantage of hardware parallelism, FPGAs exceed the computing power of digital signal processors (DSPs) by breaking the paradigm of sequential execution and accomplishing more per clock cycle. BDTI, a noted analyst and benchmarking firm, released benchmarks showing how FPGAs can deliver many times the processing power per dollar of a DSP solution in some applications.[5] Controlling inputs and outputs (I/O) at the hardware level provides faster response times and specialized functionality to closely match application requirements.

4.2 Time to market

FPGA technology offers flexibility and rapid prototyping capabilities in the face of increased time-to-market concerns. You can test an idea or concept and verify it in hardware without going through the long fabrication process of custom ASIC design.[6] You can then implement incremental changes and iterate on an FPGA design within hours instead of weeks. Commercial off-the-shelf (COTS) hardware is also available with different types of I/O already connected to a user-programmable FPGA chip. The growing availability of high-level software tools decreases the learning curve with layers of abstraction and often offers valuable IP cores (prebuilt functions) for advanced control and signal processing.

Cost The nonrecurring engineering (NRE) expense of custom ASIC design far exceeds that of FPGA-based hardware solutions. The large initial investment in ASICs is easy to justify for OEMs shipping thousands of chips per year, but many end users need custom hardware functionality for the tens to hundreds of systems in development. The very nature of programmable silicon means you have no fabrication costs or long lead times for assembly. Because system requirements often change over time, the cost of making incremental changes to FPGA designs is negligible when compared to the large expense of respinning an ASIC.

4.3 Reliability

While software tools provide the programming environment, FPGA circuitry is truly a “hard” implementation of program execution. Processor-based systems often involve several layers of abstraction to help schedule tasks and share resources among multiple processes. The driver layer controls hardware resources and the OS manages memory and processor bandwidth. For any given processor core, only one instruction can execute at a time, and processor-based systems are continually at risk of time-critical tasks preempting one another. FPGAs, which do not use OSs, minimize reliability concerns with true parallel execution and deterministic hardware dedicated to every task.

4.4 Long-term maintenance

As mentioned earlier, FPGA chips are field-upgradable and do not require the time and expense involved with ASIC redesign. Digital communication protocols, for example, have specifications that can change over time, and ASIC-based interfaces may cause maintenance and forward-compatibility challenges. Being reconfigurable, FPGA chips can keep up with future modifications that might be necessary. As a product or system matures, you can make functional enhancements without spending time redesigning hardware or modifying the board layout.

5 The Architecture of FPGA

The typical FPGA consists of the following components:

1. Programmable Logic blocks
2. Interconnection Resources
3. Input output blocks

The general schematic of an FPGA is as shown in the figure :

5.1 Programmable Logic Block

The programmable logic block in a typical FPGA consists of Configurable Logic Blocks (CLB). The CLB can be realized in many ways; one of them being the Look Up Table (LUT) based CLB. The LUT is a one bit wide memory location . The memory address lines are the inputs to the LUT and the one bit output is the LUT output. Thus the LUT with K-inputs acts as a 2k by 1 bit memory and the user can directly implement any k input function by programming the functions truth table into the LUT [9].

5.2 Interconnect Resources

The other most important feature that decides the performance of the FPGA and its suitability for control applications is its interconnect resources. This is because the interconnection resources allow the implementation of an entire

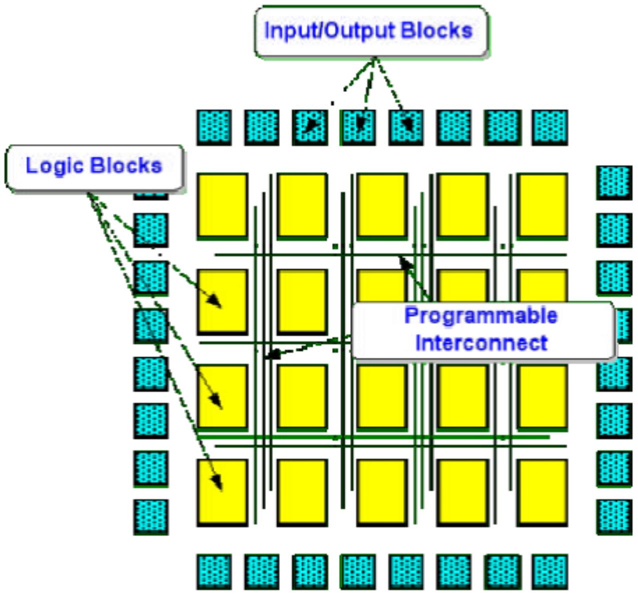


Figure 2: Basic FPGA Configuration [7]

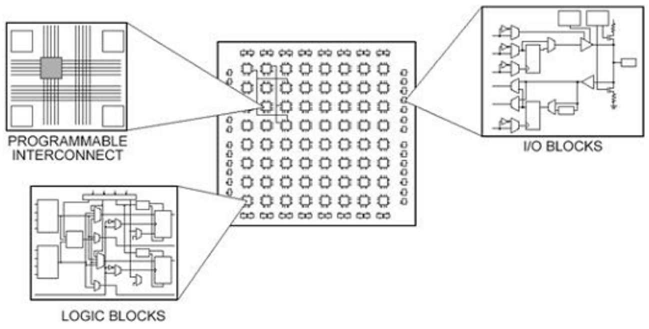


Figure 3: FPGA Schematic [8]

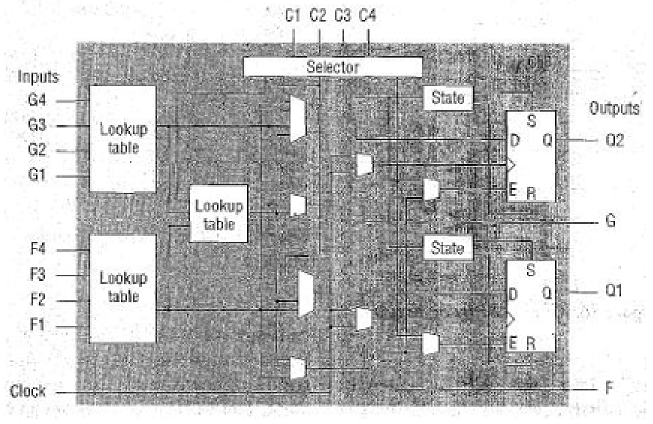


Figure 4: Xilinx FPGA-CLB Schematic [9]

digital system by providing a means of connecting various individual circuits (subsystems) that have been implemented on different CLB‘s in an FPGA.

The interconnect resources in an typical FPGA can be classified as [10] :

1. General Purpose Interconnects: Signal between CLBs and Input Output Blocks (IOBs) can be routed through switch matrices as they travel along the horizontal and vertical interconnect lines.

2. Direct Interconnects: Adjacent CLBs are interconnected directly.

3. Long Lines : Long lines provide for high fan out, low-skew distribution of signals that must travel relatively long distances. They span the entire length or width of the interconnect area. They are typically used for clock signals.

FPGA interconnects are normally unsegmented; i.e. each wiring segment spans only one logic block before it terminates in a switch box. A switch box is a switching matrix that contains programmable interconnections to all the wiring segments that terminate inside it. By turning on some of the programmable switches within a switch box, longer paths can be constructed [11].

Figure 5 shows a typical FPGA interconnection scheme.

5.3 Input Output Blocks (IOB)

The IOB provides the interface between the FPGA and the real world signals. The IOB consists broadly of I/O pads. The I/O pads connect to one of the pins on the IC package so that the external signals can be input to or output from the array of logic cells. It also consists of tristate buffers, which enable the signals to be input to and output from the logic array.

Flip flops are provided so that the input and the output values can be stored within the IOB. Each IOB has also got a variety of other features like re programmability of the input threshold to respond to either TTL or CMOS logic levels. The FPGA can be a fine grained or a coarse grained device. A fine grained FPGA consists of a large number of small width programmable logic

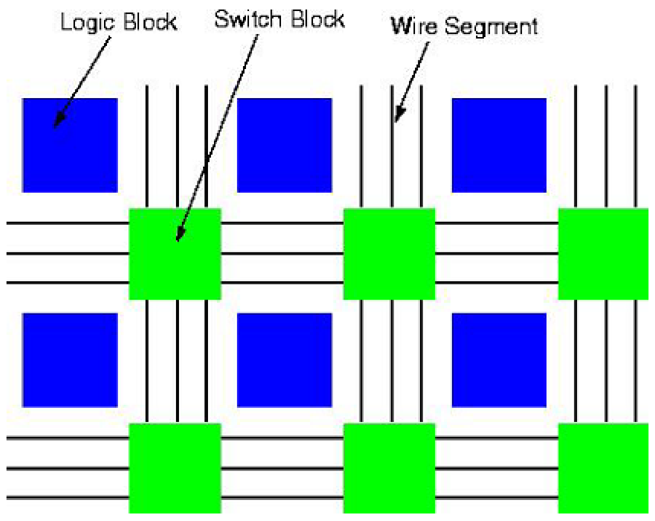


Figure 5: FPGA Interconnection schematic [11]

resources that can be used to implement a variety of functions. A typical example of such an FPGA would be the Atmel AT40K.

6 FPGA PROGRAMMING

Programming the FPGAs are necessary in order to make them ready for use. (ISE Compiler for Xilinx, and Quartus II for Altera are used for this purpose.)

Two methods are used to program the FPGA.

(1) Graphical Design

(2) Hardware Description Language (HDL)

6.1 VHDL

VHDL stand for Very High–Speed Integrated Hardware Description Language

VHDL is one of the most popular hardware description languages. With VHDL, you can program the FPGA and simulate the program before downloading it to the FPGA.

In VHDL,There are two types of codes:

(1)Synthesizable Codes: are the codes which can be implemented as hardware on FPGAs. All VHDL codes are not synthesizable. Therefore FPGA designers shall be carefull while choosing the codes.

(2)Non-synthesizable Codes: are used for simulation purposes and can not be implemented on FPGAs.

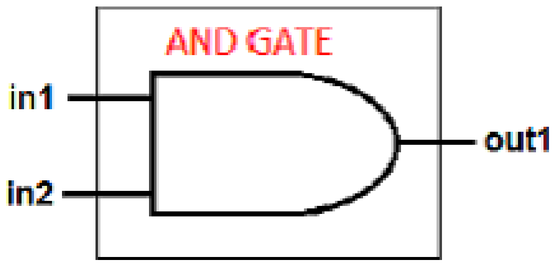


Figure 6: AND gate

```
library IEEE;                     -- Name of the library
use IEEE.STD_LOGIC_1164.ALL; -- name of the package in IEEE library
```

Figure 7: Packeges

6.2 Design units of VHDL

VHDL has four design units:

(1) Packages: A collection of declarations that can be used in more than one design.

(2) Entity: All inputs and outputs of the module are defined in this part.

(3) Architecture: Functions between inputs and outputs are defined in this part.

(4) Configurations: Architecture and Entity are associated.

7 Example of FPGA programming

We will design a 2-input AND GATE module

What will we do in this project?

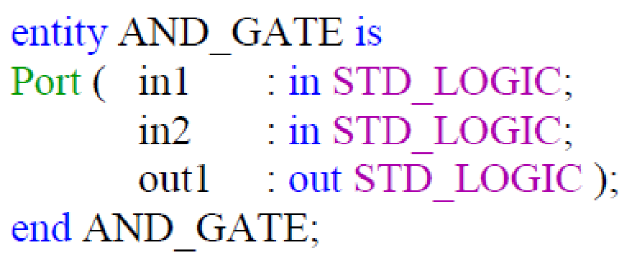
- Write the program code with VHDL
- Create A Test Bench / Debug and Verify The Code
- Simulate the project.

7.1 Write the program code with VHDL

(1) Packages:

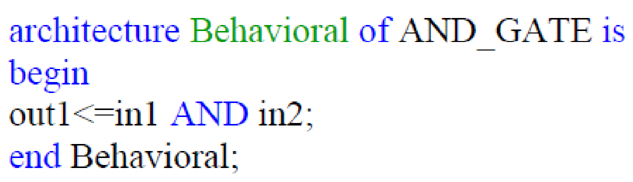
First of all, we will write the names of libraries and packages that we will need in our project. (Figure:7)

(2) Entity:



```
entity AND_GATE is
Port (   in1        : in STD_LOGIC;
         in2        : in STD_LOGIC;
         out1       : out STD_LOGIC );
end AND_GATE;
```

Figure 8: Entity



```
architecture Behavioral of AND_GATE is
begin
out1<=in1 AND in2;
end Behavioral;
```

Figure 9: Architecture

Entity defines input and output ports of the model that provide an interface to the outside world.(Figure:8)

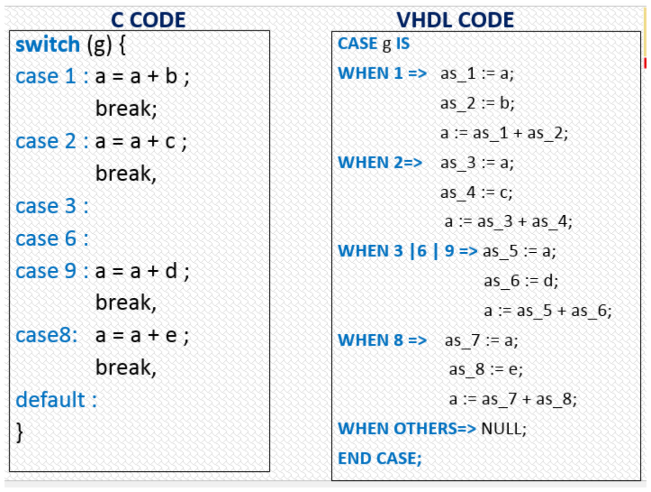
(3) Architecture: We must define the relation and function between the inputs and the output. To do this we must define an architecture.(Figure:9)

8 High Level Sysnthesis Tools

8.1 FPGAC

FpgaC is an amazingly powerful tool, even in it's current subset C form, for exploring reconfigurable computing projects. This is very useful for many C programmers wishing to develop robotics or other hobby applications. Code testing can largely be done with a traditional compiler and source code debug environment, and once working, moved to the FPGA target platform with FpgaC. Much easier than learning VHDL or Verilog, and debugging in a simulator environment with test vectors for applications which involve complex algorithms and state machines.

FpgaC is a compiler for a subset of the C programming language, which produces digital circuits that will execute the compiled programs. The circuits may use FPGAs or CPLDs as the target processor for reconfigurable computing. In short, FPGAC compiles .c code file and generates the VHDL code file. The VHDL code is used to program the FPGA.


C CODE
VHDL CODE

```
switch (g) {
case 1 : a = a + b ;
         break;
case 2 : a = a + c ;
         break,
case 3 :
case 6 :
case 9 : a = a + d ;
         break,
case8:   a = a + e ;
         break,
default :
}
```

```
CASE g IS
WHEN 1 =>   as_1 := a;
            as_2 := b;
            a := as_1 + as_2;
WHEN 2=>    as_3 := a;
            as_4 := c;
             a := as_3 + as_4;
WHEN 3 |6 | 9 => as_5 := a;
                 as_6 := d;
                 a := as_5 + as_6;
WHEN 8 =>    as_7 := a;
              as_8 := e;
              a := as_7 + as_8;
WHEN OTHERS=> NULL;
END CASE;
```

Figure 10: C code to CHDL code

8.2 Impulse C

Impulse C is a subset of the C programming language combined with a C-compatible function library

The High-level synthesis tool CoDeveloper includes an Impulse C compiler and related function library intended for development of FPGA-based applications. Impulse C is compatible with standard ANSI C, allowing standard C tools to be used for designing and debugging applications targeting FPGAs. The Impulse C compiler accepts a subset of C and generates FPGA hardware in the form of Hardware Description Language (HDL) files. Impulse C allows embedded systems designers and software programmers to target FPGA devices for C-language application acceleration.

Applications:–

Impulse C is used for applications including image processing and digital signal processing on embedded systems, as well as for acceleration of high-performance computing applications including financial analytics, bioinformatics and scientific computing.

Target platforms:–

Impulse C supports FPGAs from Xilinx and Altera,

Impulse CoDeveloper C-to-FPGA Tools

Impulse C$^{\text{TM}}$ allows you to compile C-language directly into optimized logic ready for use with popular FPGA devices. Use the Impulse tools to quickly prototype mixed software/hardware systems and perform design iterations in just minutes or hours, instead of days or weeks.

9 Conclusion

The adoption of FPGA technology continues to increase as higher-level tools evolve to deliver the benefits of reprogrammable silicon to engineers and scientists at all levels of expertise.

This report has explored the introduction of FPGA, Features and advantages of FPGA technology, and the world of pre-fabricated FPGA architectures. While these devices have changed dramatically in last two decades, it is clear that many fundamental questions remain, driven by rapid changes in technology and applications

10 References

[1] The Linley Group, A Guide to FPGAs for Communications, First Edition (July 2009).

[2] Fpga Article: http://EzineArticles.com/4545291

[3] Fpag :http://en.wikipedia.org/wiki/Fpga

[4] Hardware descriptor language: http://en.wikipedia.org/hdl

[5] FPGAs for DSP (BDTI Industry Report), 2nd ed. (Berkeley Design Technology Inc., 2006).

[6] M. Thompson, "FPGAs Accelerate Time to Market for Industrial Designs," EE Times (July 2,2004). http://www.us.design-reuse.com/articles/8190/fpgas-accelerate-time-to-market-for-industrial-designs.html.

[7] trugramer: http://www.coe.montana.edu/ee/courses/-ee/ee367/pdffiles/truegamer.pdf

[8] National Instruments: http://www.ni.com. FPGA based control: Millions of transistors at your command, 2004.

[9] Stephen Brown. FPGA and CPLD architectures: A Tutorial. IEEE Design And Test Of Computers, 1996.

[10] Charles H Roth Jr. Digital System Design Using VHDL. Brooks/Cole, 1998.

[11] Wikipedia: http://www.wikipedia.org. Field programmable gate array, 2005.