# Field Programmable Gate Array (FPGA)

Christian Baumann (christian.baumann@student.uibk.ac.at)

Summary paper for the seminar "Embedded System Architecture"
University of Innsbruck
January 13, 2010

## 1 Introduction

Field-Programmable Gate Arrays (FPGAs) are pre-fabricated silicon devices that can be electrically programmed to become almost any kind of digital circuit or system [2]. They have many advantages over Application Specific Integrated Circuits (ASIC). ASICs are designed for specific application using CAD tools and fabricated at a foundry. Developing an ASIC takes very much time and is expensive. Furthermore, it is not possible to correct errors after fabrication. In contrast to ASICs, FPGAs are configured after fabrication and they also can be reconfigured. This is done with a hardware description language (HDL) which is compiled to a bit stream and downloaded to the FPGA. The disadvantages of FPGAs are that the same application needs more space (transistors) on chip and the application runs slower on a FPGA as modern as the ASIC counterpart. Due to the increase of transistor density FPGA were getting more powerful over the years. On the other hand the development of ASICs was getting slower and more expensive. Therefore FPGAs are increasingly applied to high performance embedded systems.

## 2 FPGA Structures

### 2.1 Basic Structure

In the chapter the basic structure of a FPGA will be described. Xilinx is one of the biggest FPGA manufacturer. A Xilinx FPGA is made up of three basic blocks:

- CLB: The Configurable logic blocks are were the user specific functions are calculated.
- IOB: The Input/Output block make it possible to connect the FPGA to the other elements of the application
- Interconnect: Interconnect is essential for writing between CLB and from IOBs to CLBs.

The CLBs are located at the center of the chip and the IOBs on the periphery. The interconnect is necessary to implement several designs on the FPGA. The distributed configuration memory controls the behavior of the CLBs and IOBs by storing the program. Next the implementations of CLBs, interconnect and IOBs are described more in detail.

**CLB** Figure 1 shows a simplified CLB of a Xilinx FPGA. A CLB is used to implement custom combinational or sequential logic. It is composed of a lookup table (LUT) controlled by 4 inputs to implement combinational logic and a D-Flip-Flop for sequential logic. A MUX is used to select between using the output of the combinational logic directly and using the output of the Flip-Flop. One CLB is programmed by downloading the truth table of the logical function to the LUT (16 bit) and the control bit of the MUX (1 bit). By using multiple copies of the this structure any combinational and sequential logic circuit can be implemented.

The figure only shows a very simplifies version of a CLB slice. In real a CLB is considerable more complicated. To speed up additions extra logic for a carry chain is provided. Furthermore,

additional MUX are provided, so that the D-Flip-Flop can bis user without and in conjunction with the LUT. Additionally the LUT can also be used as memory.

Once the CLB slices have been configured to implement logical functions they have to be connected to implement bigger logical function. This is realized by programmable interconnect points (PIP) showed at the right side in figure 1. It is a pass through transistor. The gate of the transistor is connected to the memory. If that memory bit is set to one the ends of the transistor are logically connected, otherwise no connection is made. Therefore different connection can be achieved by loading the memory.

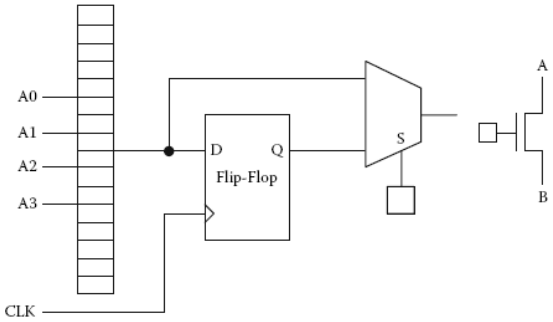


Figure 1: Configurable logic block (CLB) [1]

**Interconnect** Multiple copies of CLB slices are arranged in a matrix on the surface of the chip. The CLBs are connected column-wise and row-wise. At the intersections of columns and rows are programmable switch matrices (PSM). This can be seen in figure 2. In this figure the output of one CLB is connected with the inputs of two other CLBs. The signal passes through three PIPs and two PSMs. While the PSMs make the FPGA versatile, they slow down the signals. Therefore FPGA implementation become considerable slower then their ASIC counterparts. Therefore FPGA designers added many other interconnect in addition to PIPs and PSMs. Some architectures implement nearest neighbor routs. There are also routes that skip some PSM. Furthermore, there are extra routes for e.g. reset lines or clock lines.

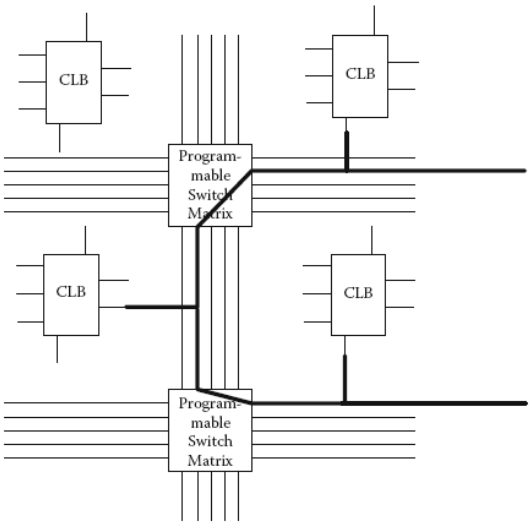


Figure 2: Programmable interconnect [1]

**IOB** Input/output blocks are used to get the signals into the FPGA and out of the FPGA. Figure 3 shows a simplified IOB. Each IOB can be used as and input and output depending on the state of the output enable (OE). If OE is set to one the IOB acts as an output, otherwise as an input. The OE bit can be programmed statically or set to the output of a CLB. IOBs contain D-Flip-Flops for

latching the input and output signals. The latches can be bypassed by appropriately programmed MUXs.

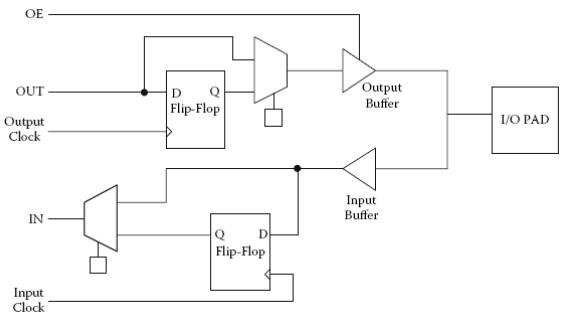


Figure 3: Input/Output block

Additionally to the described basic block the FPGA has a distributed configuration memory. The memory locations are distributed over the chip and need to be loaded to configure the FPGA. Special I/O pins that are not configurable by the user are used to load the program to the memory.

## 2.2 Modern FPGAs

In the previews section the basic building blocks of a FPGA were described. Additionally to these blocks modern FPGAs have additional units that make the design of applications easier and more efficient. Small memories and arithmetic units are difficult to implement on CLBs. Therefore modern FPGAs provide embedded memories and embedded logic blocks for arithmetic calculations. The most common arithmetic calculation is the multiplication, but many other operations can be provided. The advantage of embedded logic blocks are better speed and space. Additionally embedded memories are easier to interface than extern memories. DSP applications are often good targets for implementation on FPGA. Thus manufacturer add embedded block to be useful for implementing DSP functions, e.g. multipliers. Furthermore, they provide DSP logic designated for streaming data applications. FPGA often communicate with microprocessors. Because of that reason, embedded processor cores are added to many FPGAs. The main advantage of embedded microprocessors is the reduction of the latency of communication between microprocessor and the FPGA. The Xilinx 4 family has support for additional operations configured by the designer and implemented by CLBs with th auxiliary processing unit (APU) interface. In contrast to embedded processors, soft cores are build directly on the FPGA fabric. The advantages are that they are configurable and the clock can be the same as that of the FPGA. Furthermore, soft processor cores are easier to interface. The big disadvantage is the slower clock rate.

### 2.2.1 The Xilinx Virtex-6 FPGA family

Virtex-6 the newest FPGA family from Xilinx. It is divided into the LXT, SXT and HXT sub-family. Each sub-family contains a different ratio of features to address the needs of many logic designs:

- Virtex-6 LXT FPGAs: High-performance logic with advanced serial connectivity
- Virtex-6 SXT FPGAs: Highest signal processing capability with advanced serial connectivity
- Virtex-6 HXT FPGAs: Highest bandwidth serial connectivity

CLBs possess a LUT which can be configured as one 6-input LUT or two 5-input LUTs. The LUT also can be used as 64 bit RAM or 2 32 bit RAMs.

Every Virtex-6 FPGA has between 156 and 1064 dual-port block RAMs, each storing 36 Kbits. Each block RAM has two completely independent ports that share nothing but the stored data.

All Virtex-6 FPGAs have many dedicated, full-custom, low-power DSP slices combining high speed with small size, while retaining system design flexibility.

Each DSP48E1 slice fundamentally consists of a dedicated 25  18 bit two's complement multiplier and a 48-bit accumulator, both capable of operating at 600 MHz. The multiplier can be dynamically bypassed, and two 48-bit inputs can feed a single-instruction-multiple-data (SIMD) arithmetic unit (dual 24-bit

add/subtract/accumulate or quad 12-bit add/subtract/accumulate), or a logic unit that can generate any one of 10 different logic functions of the two operands. The slice provides extensive pipelining and extension capabilities that enhance speed and efficiency of many applications. Every FPGA of the family contains a System Monitor circuit providing thermal and power supply status information. Sensor outputs are digitized by a 10-bit analog-to-digital converter (ADC). This fully tested ADC can also be used to digitize up to 17 external analog input channels. All but one Virtex-6 device has between 8 to 72 gigabit transceiver circuits. Each GTX transceiver is a combined transmitter and receiver capable of operating at a data rate between 155 Mb/s and 6.5 Gb/s. An integrated Ethernet MAC block is easily connected to the FPGA logic, the GTX transceivers, and the SelectIO resources. All but one FPGAs of the Virtex-6 family include at least one integrated interface block for PCI Express technology.

The **XC6VHX5651** is an advanced FPGA of the Virtex-6 family. It consists of 566784 logic cells and 88560 CLB slices with maximum 6370 bytes distributed RAM. The FPGA contains 864 DSP48E1 slices and has 1824 18kb RAM blocks and 912 36kb RAM blocks. That is a total memory of 32832kb.

# 3 Configuring FPGAs

FPGAs are not programmed directly. Synthesis tools translate the code into bit stream, which is downloaded to the configuration memory of the FPGA. Commonly, hardware description languages (HDL) are use to configure the device. But resent trends also offer the possibility to high level languages. Furthermore, there are library based solution which are optimized for a specific device.

**Hardware description language** Using a HDL is the most common approach to configure a FPGA. There are two dominating languages, VHDL and Verilog. Both languages have the power of international standards and working groups behind and are similar powerful. VHDL was developed in the 1980s by the Department of Defense and is essentially as subset of ADA with extensions for describing hardware. Verilog was originally a C-like programming language to model hardware and later became IEEE standard like VHDL. The languages support different levels of abstraction, but most configurations are done at the register transfer level (RTL). The design resembles soft development more than hardware development, but there are big differences. Software programs have a sequential execution model and the correctness of the program depends on the sequential executed commands. Decision points are very common. Furthermore, programmer does not have to care about data flow between registers and memory. Hardware designs consists of several block of hardware running in parallel. The designer tries to avoid decision points, because of performance reasons. The wires for data movement have to be explicitly written on the FPGA. To configure with a FPGA with HDL a developer needs two programs. A tool that allows the programmer to test and simulate a configuration on a PC and the synthesis tool that is needed to convert the HDL program into a bit stream and download it to the FPGA.

**High level language** There are also approaches using high level languages that make designing FPGA application more alike software development.

SystemC is a C++ library that allows to specify and simulate hardware processes using a C++ syntax.

Handel-C is an extended subset of ANSI C that allows developer to specify their designs with C. It can be synthesized directly for implementation on FPGAs.

With Accelchip it is possible to generate VHDL or Verilog code block for common MATLAB DSP functions.

**Library-based solutions** There functions that are needed in many FPGA design. For those functions are already optimized solutions available. Therefore the FPGA manufacturer Xilinx and Altera offer parameterized macros to generate code for common blocks such as arithmetic functions or specialized memories. The output of the macros is HDL code that can be included in the developers synthesis process. This reduces the development time.

# 4 An FPGA implementation of a flexible, parallel image processing architecture [3]

A common FPGA-application is a embedded vision system such as vehicle detection and security systems. After acquisition of the image, a significant amount of computation is require in the pre-processing phase

before the features are extracted and classified. These computation contain fine-grained parallelism that makes any sequential algorithm inefficient. Therefore an efficient solution is to use an array of preprocessing elements. In this example these pre-processing element are implemented on a Xilinx Virtex-2 FPGA. Figure 4 shows the image pre-processor.

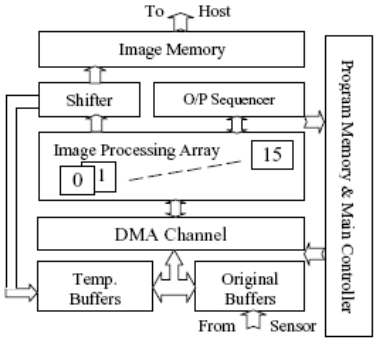


Figure 4: Image Pre-Processor

The source frame may either be the output of the sensor, or a temporary buffer to which a previous output of the image pre-processing layer was stored. The DMA channel then distributes the source pixels to the Image Processing Array. The array consists of 16 elements. The pre-processed image resulting from the parallel array is loaded into an image memory accessible to the host processor, which is responsible for feature extraction and classification. The image pre-processing architecture communicates with its host through a program memory. The host processor awaits feedback through status reporting from the main controller before picking up data from the image memory. The shifter can be configured to compensate for scaling factors or simply to normalize the output of the array. An output sequencer multiplexes between the outputs of the processing elements and provides the handshake signals necessary to confirm data delivery.

Figure 5 shows one of the 16 pre-processing elements (PPE) of the array showed in figure 4. Each element is implemented by using DSPs and CLBs on the FPGA. All 16 processing elements working in parallel and can be seen as a processor specialized for image pre-processing.It has an 16 bit input data-path, 32 bit output data-path and a RISC-like instruction-set. The element operates on 16 bit data and produces a 32 bit output. For this purpose the advantage of a FPGA is the not fixed word-size. The element receives instructions form the main-controller and operates on pixels stored in local memory. A PPE also contains a small memory for coefficients, convolution and correlation masks as well as matrix constants. Furthermore, there are two 16 bit registers for temporary storage. Once the computation is complete the output sequencer gets informed that the result is available at the output.

Figure 6 shows the complete vision system.

The a architecture was implemented using a Celoxica RC1000-PP board with a Xilinx Virtex FPGA. Images are taken with as 256x256-pixel sensor connected to the board via external I/O. The board communicates with the host via a PCI bus.

The architecture brings a performance optimization compared with a software implementation executed

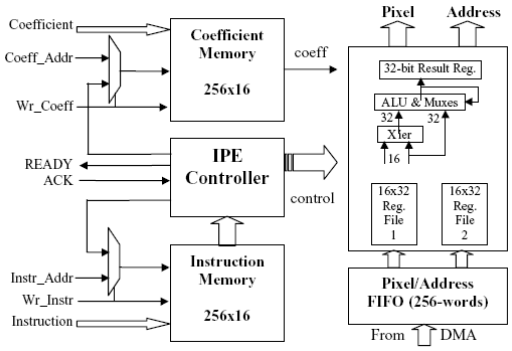


Figure 5: Pre-processing element

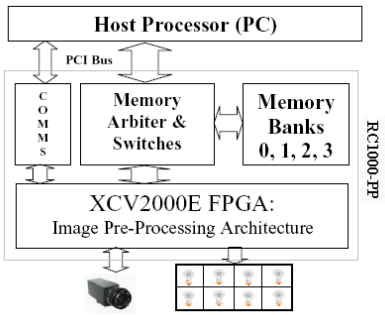


Figure 6: Complete vision system

on a PC. At 50 MHz clock frequency the parallel processor implementation on the FPGA can achieve a throughput of up to 125 Frames/s, whereas the original software application which normally runs on a standard PC achieves 50 Frames/s with a processor clock frequency of 266 MHz. This is mainly due to the parallelism of the architecture.

# 5 Conclusion

This paper gave a short overview on Field programmable Gate Arrays (FPGA). The first section described the basic internal blocks and circuities of FPGAs. Additionally as an example for a modern FPGA family the Xilinx Virtex-6 family was described shortly. The second part was dedicated the configuration of FPGAs. An overview of the available tools and possibilities was given. Finally, a practical FPGA application was described, a flexible and parallel image processing application.

# References

[1] M. Michael Vai David R. Martinez, Robert A. Bond. *High Performance Embedded Computing Handbook*. CRC Press, 2008.

[2] Jonathan Rose Ian Kuon, Russel Tessier. Fpga architecture: Survey and challenges. *Foundations and Trends in Electronic Design Automation*, 2007.

[3] Peter Lee Stephanie McBader. An fpga implementation of a flexible, parallel image processing architecture suitable for embedded vision system. *IEEE*, 2003.