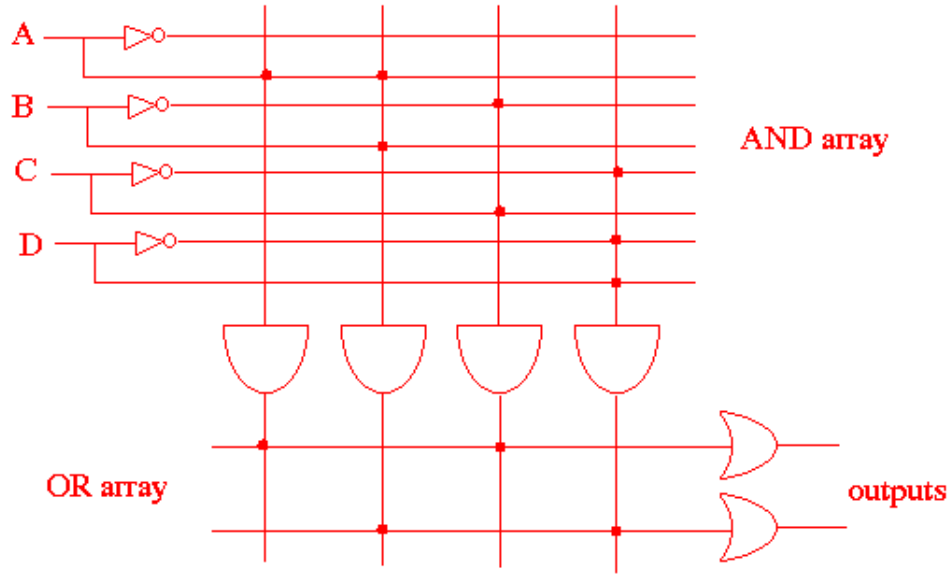
Introduction to FPGAs

From PLA to FPGA

An FPGA, like other programmable devices, can make it easy to implement complex logic circuits. Creating intricate circuit designs from discrete parts, such as TTL chips, can be very tedious and error prone. It can take a large number of chips to create a design of only moderate complexity. It takes a lot of time to wire together a large number of chips. Additionally, it can often be difficult to find misplaced wires when debugging a complex circuit. With a programmable device, such as an FPGA, it is possible to describe the design in a program written in a hardware description language (HDL). The desired functionality of the device is described within the HDL. Then, this program can be downloaded to the programmable device. If the device does not function as it should, it is only necessary to debug the program as opposed to debugging the wiring of a circuit made from discrete chips. As a result, it is possible to implement a complex logic design in a manner which is easy to test, debug and even change.

Programmable devices began as small PLAs, PALs and PLDs, but as the demand for programmable chips increased these smaller devices evolved into larger ones such as CPLDs and FPGAs. The first programmable chips to be developed were *programmable logic arrays*, PLAs. These devices were created for use as a better method by which to create large logic circuits.

The following picture depicts the architecture of a basic PLA.

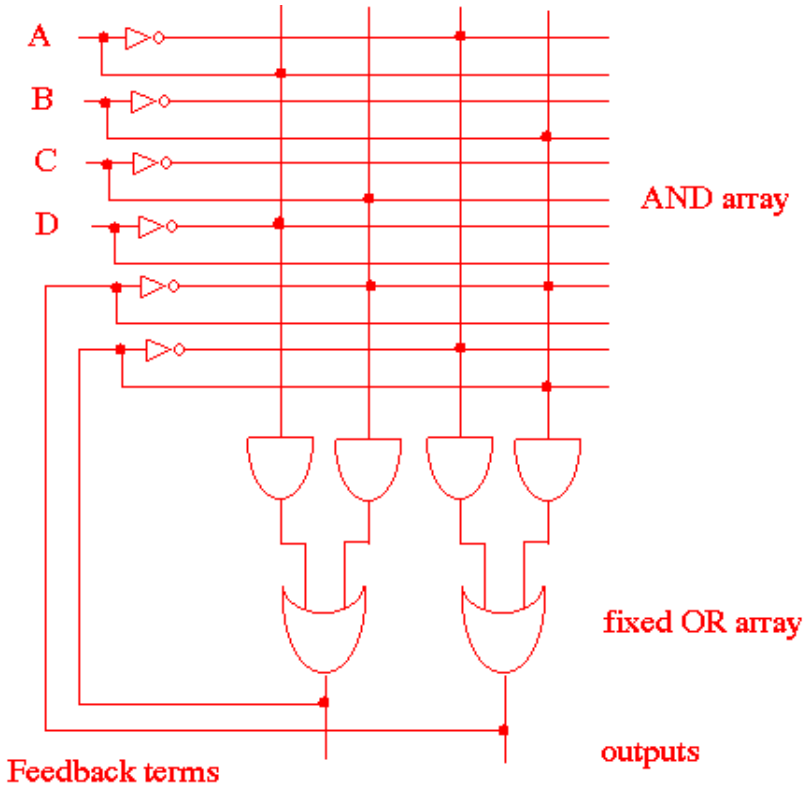


A PLA

There are several key components to the architecture of this PLA. The first step toward implementing a complex logic function is to create the product terms. To form these product terms for the desired logic function, the inputs and their compliments are sent into an array of "anded" signals. The outputs of the "and" array, the product terms, become the inputs to an array of "or" gates. The sum of products representation of the desired logic function is formed as a result of the passing the product terms through the "or" array. It is important to note that a sum of products equations can be forned for each of the individual outputs of the device.

The PLA gave the user a lot of design flexibilty because of the fact that it has both a programmable "and" array and a programmable "or" array. However, this flexibility was not always utilized by the logic designer. As a result a simpler programmable device evolved from the PLA. These programmable chips were known as *programmable array logic devices*, PALs.

The architecture of the a basic PAL is pictured below.

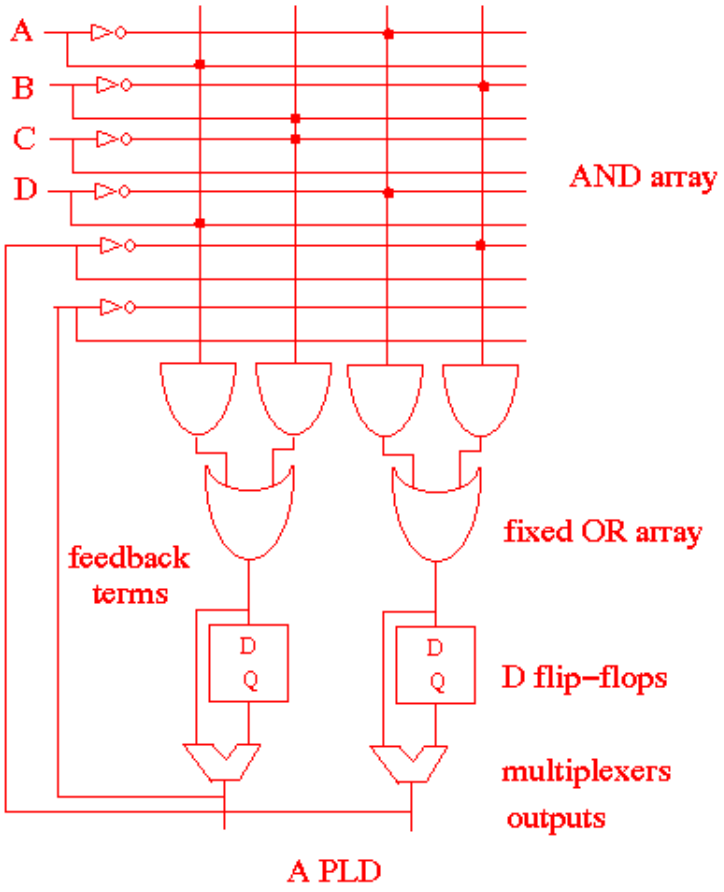


A PAL

The architecture of this device is very similar in nature to the architecture of the original PLA, however there are several key differences. First, the programmable "or" array has been replaced with a set of fixed connections to the or gates. The PAL is simpler than the PLA because only the connection to the "and" array can be programmed, whereas the connections to both the "and" and the "or" arrays can be specified for the PLA. The second change in the architecture present in the PAL is the addition of the feedback terms. The outputs of the fixed "or" array are fed back to some of the inputs of the "and" array. Because of this feedback, it is possible to use the results of the sum of products as inputs in order to create more complex, multi-level logic designs.

The PLAs and PALs were both good types of devices for creating logic circuits which were purley combinational, however they can not be used to do sequential logic circuits without the addition of external flip-flops. So, the next logical step in the evolutionary process of programmable logic devices was to add internal filp-flops to the existing architecture of the PALs to create *programmable logic devices*, PLDs.

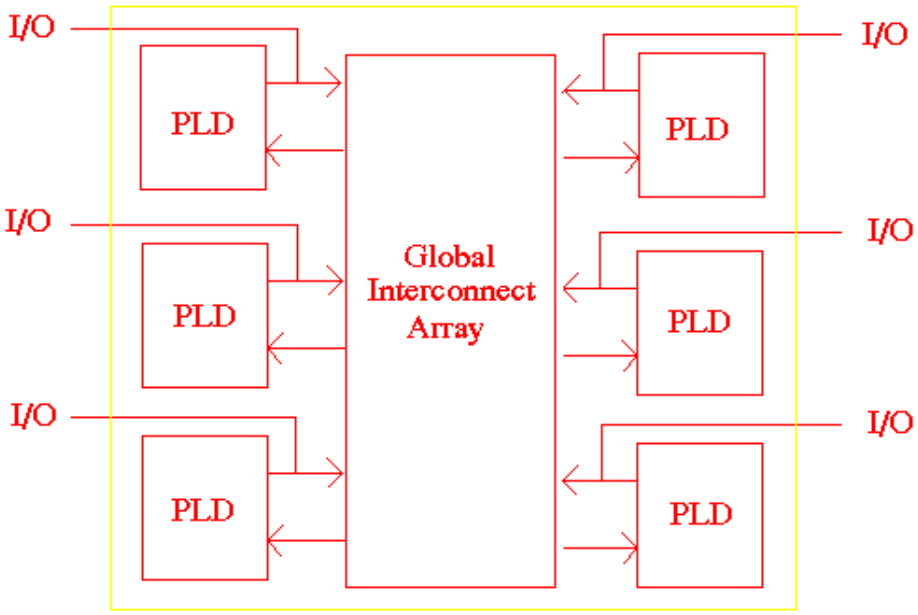
The picture below demonstrates the architecture of a simple PLD.



A PLD

Again, the architecture of a PLD is very similar to both the architecture of a PLA and the architecture of a PAL. The feedback terms still exist so that the user can create multi-level designs. However, the user now has the option to choose between the output of the flip-flops or the output of the purely combinational portion of the circuitry. The multiplexers, which are present in the architecture of a PLD, allow the user to make this choice and select the combinational or the sequential output. In fact, the unique architecture of a PLD makes it possible to implement circuits which utilize both sequential and combinational logic.

As the technology surounding programmable devices improved, new devices were developed which combined several PLDs together on a single integrated circuit to form *complex programmable logic devices*, CPLDs. Basically, a CPLD consists of several blocks, each of which is a PLD, which are connected together. The I/Os of each of the PLD blocks are connected by a global interconnect array. The following diagram demonstrates the basic architecture of a CPLD.

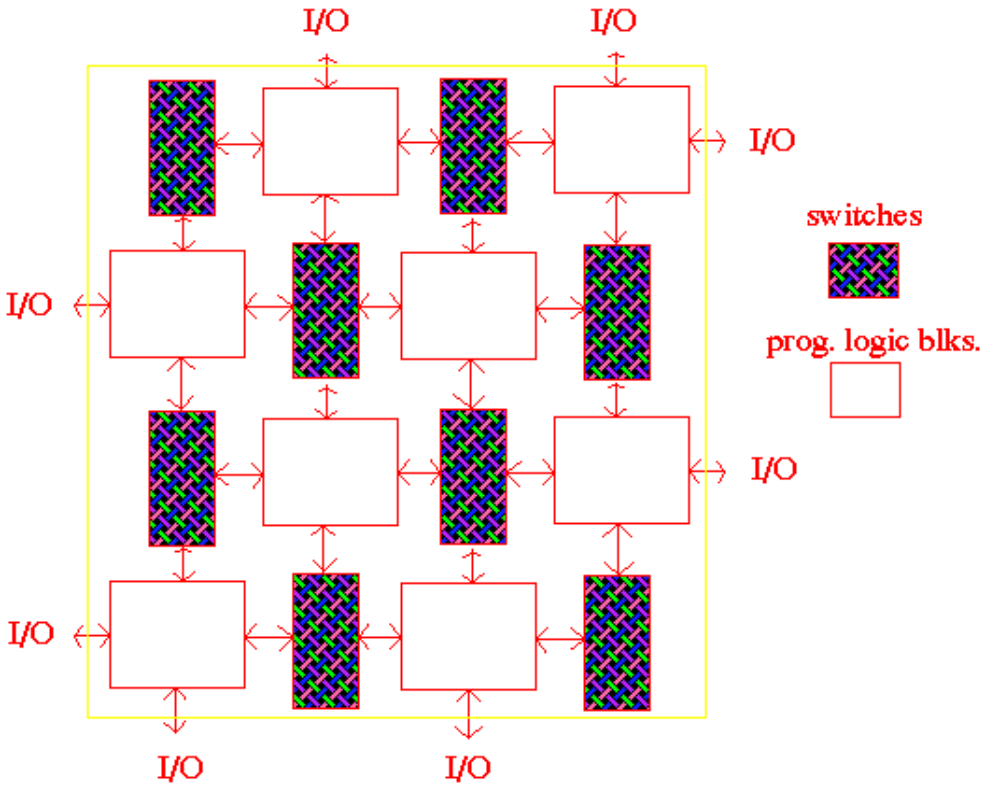


A CPLD

Because of its structure, a CPLD has two levels on which it can be programmed. Each PLD block of the CPLD can be programmed. Then, the interconnect between the individual PLD blocks can also be programmed. CPLDs are much larger devices than PLAs, PALs and PLDs, and can be used to implement large and complex logic designs.

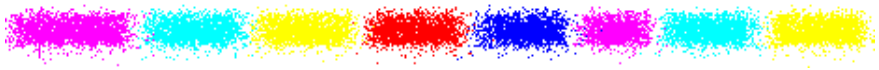
The jump from CPLDs to *field programmable gate arrays*, FPGAs, is not a very great one. In fact, some people consider CPLDs to be FPGAs. In essence, an FPGA is merely a bunch of programmable logic blocks in an array with interspersed switches that can be programmed to set the interconnection between these logic blocks. Just as with CPLDs, the FPGA has two levels of programmability. Each logic block can be individually programmed to perform simple logic functions. Then, the switches can be programmed so that the blocks are connected to implement the desired logic function.

There is not one specific type of architecture for an FPGA. Each of the companies which manufacture FPGAs utilize completely different architecture designs. The following diagram depicts, in a very general sense, the concept of an FPGA.



A FPGA

The differences between the FPGA architectures used by various companies occur in the manner in which the interconnect, i.e. the switches, and the programmable logic blocks are laid out. An FPGA could be designed as the one in the above picture, or it could be configured with all of the switches running through the middle of the device with the logic blocks situated on either the right or the left hand side. These examples are only two of many achitecture options for FPGA. But, it is important to note that the second example architecture described above, is exactly the architecture of a CPLD. Thus, it is easy to see how the same device can be cross-listed as both a CPLD and a FPGA.



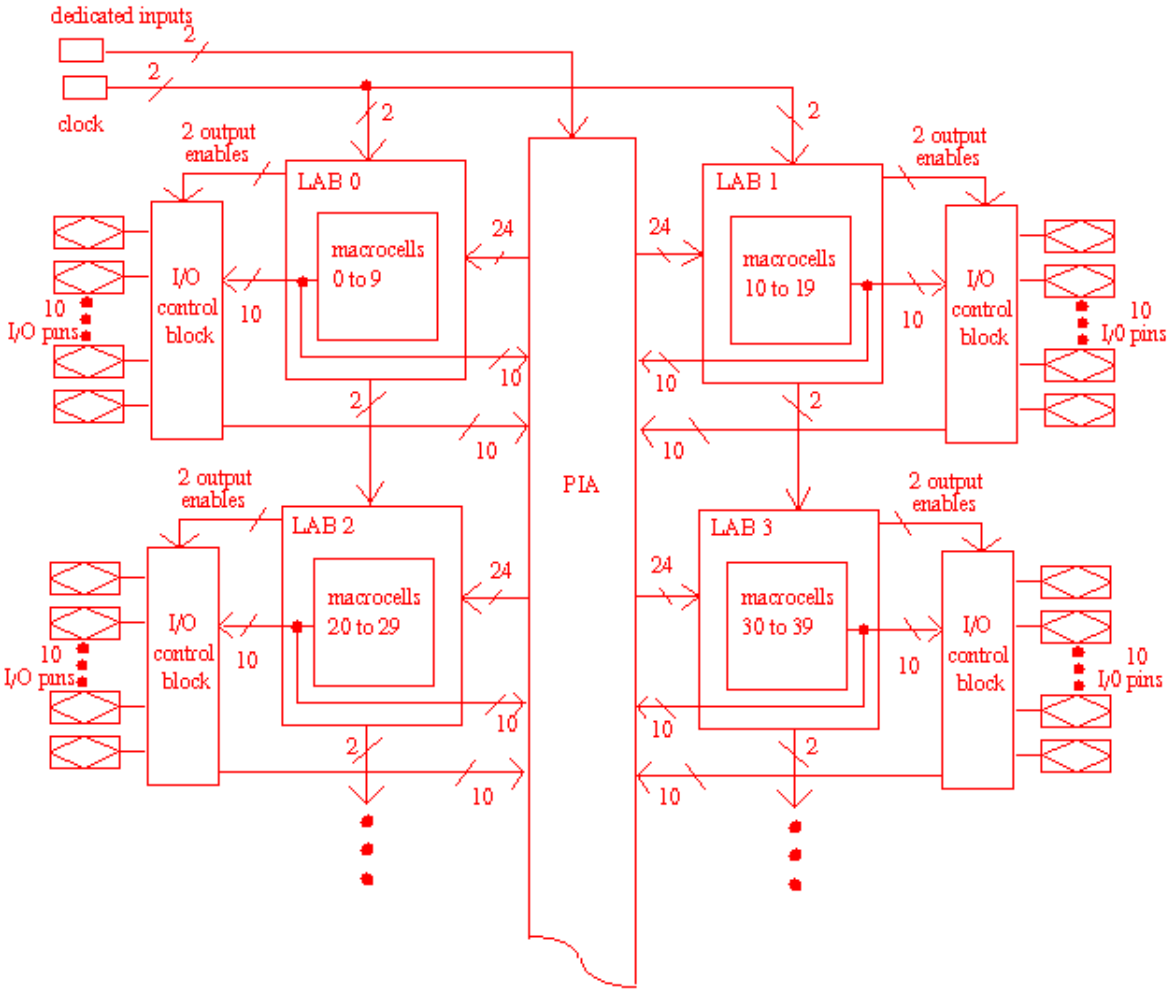
The Architecture of the NFX780_84 FPGA from Altera

The specific architecture of the NFX780_84 FPGA from Altera is similar in nature to the architecture discussed previously for a CPLD. This specific device is made up of programmable logic blocks. These blocks are called *logic array blocks*, LABs, and there are 8 of them within the device. Each one of these LABs is equivalent to 24V10 PLD. Typically, 22V10 PLDs are used in lab for digital design. Each one of these 22V10 PLDs has 22 inputs, which means that they can handle a maximum of 22 product terms. Additionally, these devices have 10 outputs, which means that there are only 10 flip-flops on the chip. Now, a single 24V10 PLD still has the same output and flip-flop characteristics as the 22V10, however it can support 24 product terms as opposed to 22. In total, there are 8 of these 24V10s on the NFX780_84, each of which is larger than a single device previously used in the digital design course.

This one chip is equivalent to more than 8 of the smaller devices. With this larger device, it is possible to implement much larger logic designs.

In addition to the LABs, the NFX780_84 contains a *programmable interconnect array*, PIA, which is similar to the global interconnect array present in a CPLD. Essentially, the PIA is a global bus which connects any signal source to any destination on the device. All of the LABs are linked via the PIA. Each LAB is made up of 10 logic units called macrocells (there is a total of 80 macrocells on the device). All dedicated I/Os which the macrocells output from the LAB feed into the PIA, and, from there, are accessible by all of the other LABs.

A diagram of the general architecture of the NFX780_84 FPGA from Altera is included below.

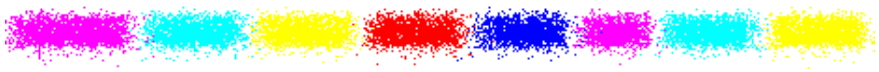


The NFX780_84

The NFX780_84 from Altera has a very unique feature in that each LAB can be configured as a 128X10 block of SRAM. The NFX780_84 is one of the only FPGAs on the market with built in SRAM which the user can enable, delcare and use. As with any industry, companies tend to try their hands at different aspects of the industry. This scenario is exactly the case for the programmable device industry.

The NFX780_84 was originally designed and manufactured by Intel. When Intel decided not to produce programmable devices anymore, Altera, a company which specializes in programmable devices, bought up all of technology which Intel had developed within this industry. So now, the NFX780_84 is being manufactured by Altera. Since its original design was not done by Altera engineers, it is a unique part with some extraordinary properties, an example of which is the ability to be configured for use as SRAM.

The user has a choice as to how he or she wants to configure each of the LABs. Each LAB can either be configured for use as a combinational/sequential logic block or as a block of SRAM. If configured as an SRAM block, each LAB contains 128 10 bit words. The option to declare SRAM internal to the FPGA enhances the logic designs for which this device can be used. For instance, it is possible to implement a small microprocessor on this FPGA without having to use an external ram chip.



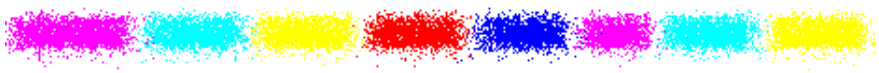
Programming the Connections in an FPGA

The connections with in the FPGA must all be made within the device after it has already been fabricated. Since the connections are internal, they are not something which we can physically wire. So, the only method by which to make these connections is to do it electrically by programming the connections.

There are several methods by which the connections can physically be made within an FPGA. One method is to make fuses where horizontal and vertical wires cross. A fuse is made up of special circuitry which allows a high voltage to be placed on selected horizontal and vertical wires. A fuse exists at every point where horizontal and vertical wires cross. The fuse is then burned out by the high voltage and the connection is opened. If a high voltage is not placed across a fuse, it remains intact and the connection remains.

Another mechanism for programming the connections in an FPGA is the anti-fuse which works on the opposite principle on which the fuse works. When a high voltage is placed across an intersection between a horizontal wire and a vertical wire, a connection between those two wires is formed.

Fuses and anti-fuses have only one major drawback. FPGAs which use these mechanisms to form connections are known as *one time programmable*, OTP, because they can not be reprogrammed. To solve this problem many FPGA manufacturers place reprogrammable switches where the fuses would normally be. In this case the switches are implemented with pass transistors, and there is static RAM on the chip which controls the pass transistor for each interconnection on the device. Loading the RAM with a 1 or a 0 controls whether the switch is open or closed. FPGAs, such as the NFX780_84, which operate in this manner are said to be *in-circuit reprogrammable*.



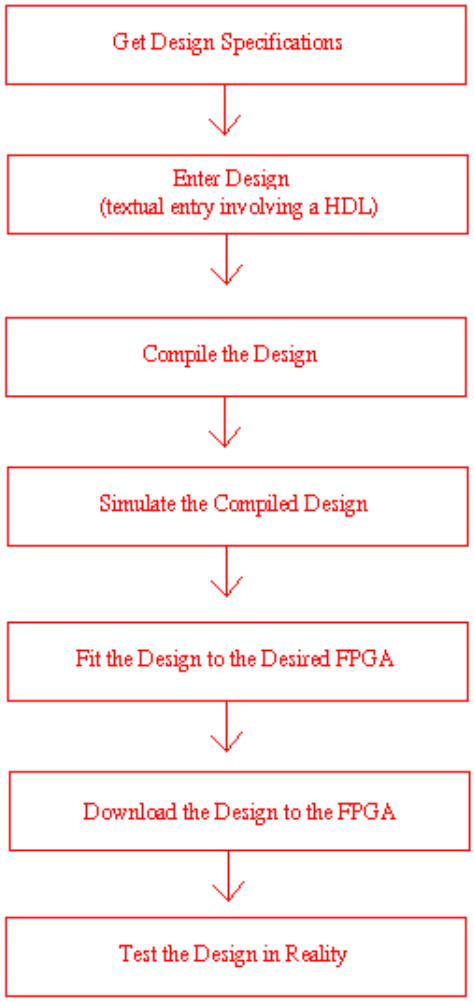
From Software to Hardware

The process of actually programming a logic design to the FPGA takes several steps. The first step is to use a software package to enter the circuit design. Many software packages have several methods by which a design can be entered. For instance, a designer may use textual design entry, in which the design is described in an *hardware description language*, HDL, such as Verilog, VHDL, ABEL or PLDasm (this language is the one which we will be using to program the NFX780_84 FPGA). Within an HDL the designer can write a behavioral model of the circuit through the use of equations which describe the functionality of the circuit. Or, the designer can describe a structural model of the circuit, such as a state-next state table, with a HDL. A HDL can be used to describe everything from simple boolean equations to complex microprocessors.

In addition to textual design entry, software packages such as maxPLUS + II from Altera offer methods of design entry which include such things as a waveform editor and a graphical editor. In the waveform editor, the designer actually creates the input and output waveforms which descibe the functionality of the circuit. In the graphical editor, the designer can choose from a list of pre-defined parts to create a circuit with the desired functions.

In addition to providing a method for design entry, the software used by designers to implement cirucits with FPGAs performs many other tasks as well. The software includes a compiler to indicate whether the circuit which has been entered is valid. It also includes a simulator which allows the designer to test the functionality of the design and see that it works correctly before the FPGA is actually programmed. Laslty, the software contains a fitter, which fits the design to the achitecture of the FPGA being used.

The description of the design provided by the designer acts as input to the software. After it is done completing the steps described above, the software produces an output file based on the logic desciption of the design. The output is a binary file which configures the switches in the FPGA so that it functions as the designed circuit.



This process is not only useful for designing circuits to be implemented with FPGAs. In fact, the same process is used in industry to design integrated circuits. The design is described in an HDL, compiled and then simulated. The only difference in the design process is that once the simulation has been verified, synthesis is performed on the design. The synthesizer takes the description and generates a transistor level implementation of the design. From this implementation, the mask layers are created and the integrated circuit is fabricated.