FPGA based Digital System Design–Issues and Challenges

Naresh Grover
Faculty of Engineering and Technology
MRIU, Faridabad

M.K. Soni
Dean
Faculty of Engineering and Technology
MRIU, Faridabad

Abstract: *Applications for Field Programmable Gate Array (FPGA) technology are substantially increasing due to its re-configurability that offers significant advantages in the flexibility. Recent advances in fabrication have increased its logic capacity as well. FPGAs are evolving at a rapid speed with improved performance and logic density. However, high power consumption in FPGAs becomes a significant factor in the design consideration. Trends in technology scaling makes leakage power a serious concern for the designers. Moreover, increased density has made these devices more susceptible to failure due to external radiations. This paper describes the generic architecture of FPGA and basic development stages for FPGA based digital design. Then it covers the issues of power consumption in FPGA and proposed strategies for its optimization and finishes with the description of radiation effects on FPGAs and the proposed mitigation techniques.*

1. INTRODUCTION

Digital designer has various options of using SSI (small-scale integrated circuits) or MSI (medium scale integrated circuits) components, simple programmable logic devices, Microprocessor/ Microcontroller, Masked Programmable Logic Devices (MPGA), CPLD, ASICs, FPGA etc. In SSI and MSI difficulties arises as design size increases and moreover interconnections grow with complexity resulting in a prolonged testing phase. Programmable Logic Devices (PLDs) include programmable array logic (PAL) and programmable logic array (PLA) wherein architecture are not scalable, power consumption and delays play an important role in extending the architecture to complex designs and implementation of larger designs leads to same difficulty as that of discrete components. The next stage of sophistication resulted in Complex PLDs (CPLDs), which were nothing else than collection of multiple PLDs with programmable interconnections. Microprocessors and microcontrollers provide a flexible computing platform and capable of executing a large class of applications. They have fixed hardware and can be programmed as per their capabilities and limitations. Application Specific Integrated Circuits (ASICs) are designed for specific applications and have fixed functionality and superior performance for a highly restricted set of applications. A quest for high capacity is met with two choices: one with Masked Programmable Logic Devices (MPGA) which is customized during fabrication, low volume expensive and has prolonged time-to-market and high financial risk and other with Field Programmable Logic Devices (FPGA) which is customized by end user, implements multi-level logic function and has fast time to market and low risk.

FPGA is a two dimensional array of customizable logic block placed in an interconnect array. It is programmable at users' site like PLD and implements thousands of gates of logic in a single device like MPGA. It employs logic and interconnect structure capable of implementing multi-level logic and is scalable in proportion with logic removing many of the size limitations of PLD derived two level architecture. Therefore it offers the benefit of both MPGAs and PLDs. It is based on the principle of functional completeness. Functionally complete elements (Logic Blocks) are placed in an interconnect framework which comprises of wire segments and switches and provide a means to interconnect logic blocks. Its interconnection framework circuits are partitioned to logic block size, mapped and routed.

FPGAs now deliver ASIC-like density and performance, while their flexibility and operational characteristics offer distinct advantages over their ASIC counterparts. As innovative architectures with embedded processors, memory blocks and Digital Signal Processors (DSPs) emerge; designers are turning more towards FPGAs for new system on chip (SoC) designs. If the design time in case of FPGA is 9 months, then it takes approximately 2-3 years in case of ASIC for the same design. Moreover, high initial ASIC cost is recovered only in very high volume products. Due to Moore's law, many ASIC market requirements are now met by FPGAs. Use of FPGAs as a % of logic market has increased from 10 to 22% in past 3-4 years. FPGAs

(or programmable logic) are the fastest growing segment of the semiconductor industry.

The architecture of various FPGAs differs from vendor to vendor and is characterized by structure and content of logic block and of routing resources. Just to examine, we will look at FPGA: Xilinx Virtex-4 and Virtex-5.

Generic Xilinx Architecture consists of:

- Symmetric Array based; Array consists of Configurable Logic Blocks (CLBs) with Look up tables (LUTs) and D-Flip-flops.
- N-input LUTs can implement any n-input Boolean function
- Array embedded within the periphery of IO blocks
- Array elements interleaved with routing resources (wire segments, switch matrix and single connection points)
- Employs SRAM technology

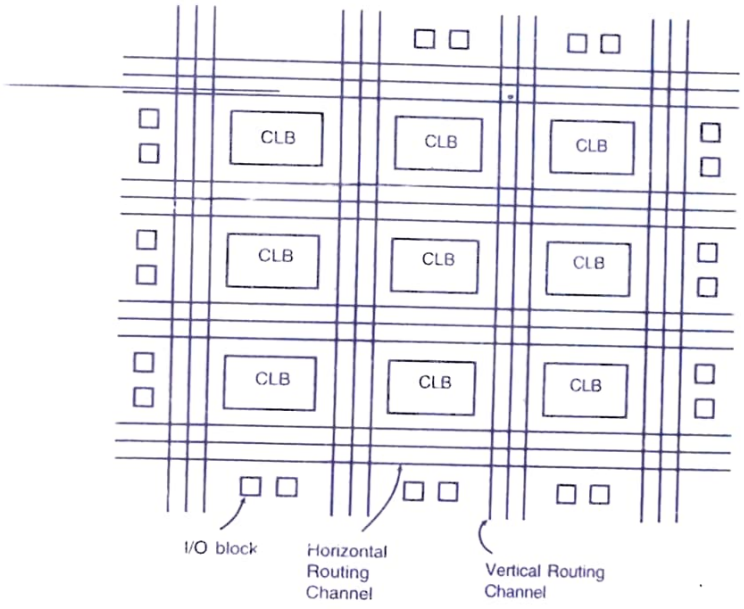

Fig.1. Generic Xilinx Architecture

The elementary programmable logic block in Xilinx FPGAs is called *slice*.[3]

The Virtex-4 FPGA slice includes:

- Two **4-input LUTs** that can implement any 4-input Boolean function, used as combinational function generators (one LUT is marked "F", the other one is marked "G").
- Two dedicated user-controlled **multiplexers** for combinational logic (MUXF5 and MUXFX). MUXF5 can be used to combine outputs of the slice's LUTs and so to implement 5-input combinational circuit. MUXFX is used to combine outputs of the other MUXF5 and MUXFX (from the other slices).
- Dedicated **arithmetic logic** (two 1-bit adders, carry chain and two dedicated AND gates for fast and efficient multiplication).
- Two **1-bit registers** that can be configured either as **flip-flops** or as **latches**. YMUX and XMUX multiplexers select the input to these registers. Note that these multiplexers aren't user-controlled: the path is selected during FPGA programming.

The simplified diagram of a Virtex-4 slice is presented below in Fig. 2.

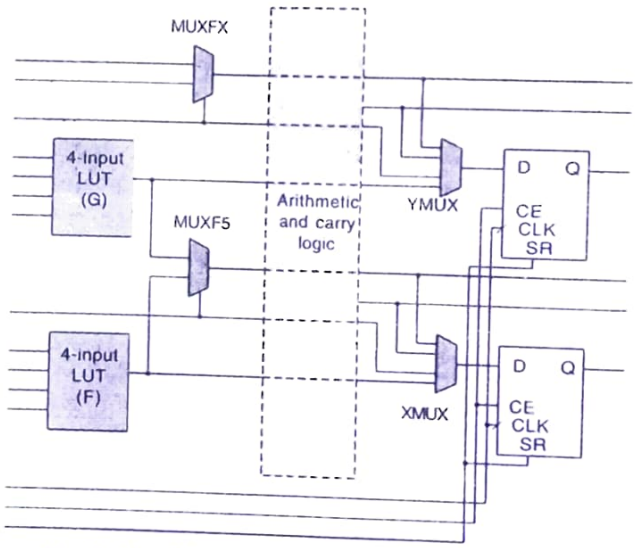

Fig. 2. Simplified diagram of a Xilinx Virtex-4 slice

The Virtex-5 slices include:

- Four **LUTs** that can be configured as 6-input LUTs with 1-bit output or 5-input LUTs with 2-bit output.
- Three dedicated user-controlled **multiplexers** for combinational logic (F7AMUX, F7BMUX and F8MUX). F7AMUX and F7BMUX combine outputs of the slice's LUTs to implement 7-input combinational circuits.

F8MUX is used to combine outputs of the F7AMUX and F7BMUX.

- Dedicated **arithmetic logic** (two 1-bit adders and a carry chain).
- Four **1-bit registers that** can be configured either as **flip-flops** or **as latches**. The input to these registers is selected by AMUX DMUX multiplexers. Note **that** these multiplexers aren't user-controlled: **the** path is selected during FPGA programming.

The simplified diagram **of a** Virtex-4 slice is presented below in Fig 3.

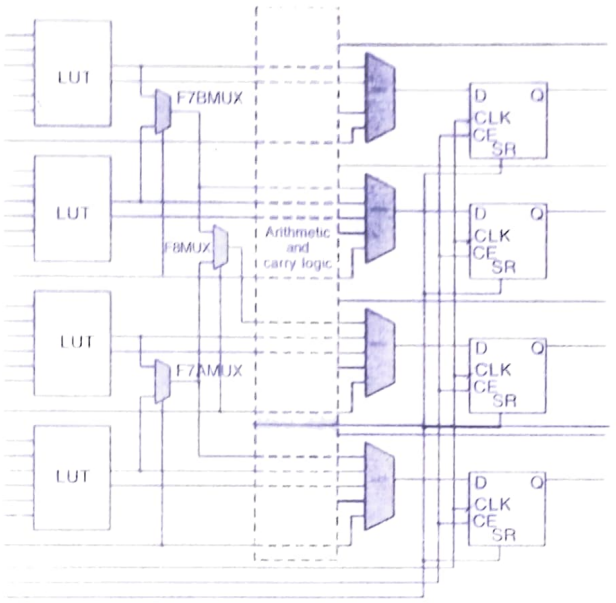

Fig. 3. Simplified diagram of a Xilinx Virtex-5 slice

The main differences from the previous Xilinx architecture are:

- Configurable 6-to-1 or 5-to-2 LUTs instead of 4-to-1 LUTs.
- 4 LUTs and 4 register bits per slice.
- Dedicated arithmetic logic circuitry doesn't include dedicated AND gate[3].

Some of the other manufacturers of programmable devices are: Quick Logic, Actel, Altera, Atmel, Dyna chip, Lucent, Motorola, Gate Field, I-cube Lattix and Aptix. Out of theses, Altera FPGAs (*Stratix* and *Cyclone* families) uses slightly different logic blocks called "Adaptive Logic Modules" (ALMs). ALM resources include:

- Two 4-input LUTs and four 3-input LUTs for combinational logic implementation.
- Dedicated arithmetic and carry logic.
- Two programmable registers.

Cyclone is the lowest cost FPGA family ($3 -$7 per chip) and includes maximum of 20K logic elements and 300Kbits of memory. Stratix is the highest density FPGA with maximum of 80K logic elements, 10Mbits memory, PLL, DSP and DDR interface blocks.

Actel produces Antifuse-FPGAs mainly for aerospace and military applications. Compared with top Xilinx and Altera devices, Actel FPGAs provide less logic resources. The basic building block for Actel flash-based FPGAs (such as *ProASIC-3*) is called *VersaTile*. Each VersaTile cell can implement any of the following:

- Any 3-input combinational logic function, OR
- D flip-flop or latch.

One VersaTile can implement only one of these, not both.

After having a broad picture of architectures of main FPGA slices, let us have the overview of the development stages/steps involved in designing FPGA.

2. DEVELOPMENT STAGES OF FPGA

Regardless of the final product, FPGA designer has to follow the following four basic FPGA development stages as shown in Fig. 4:

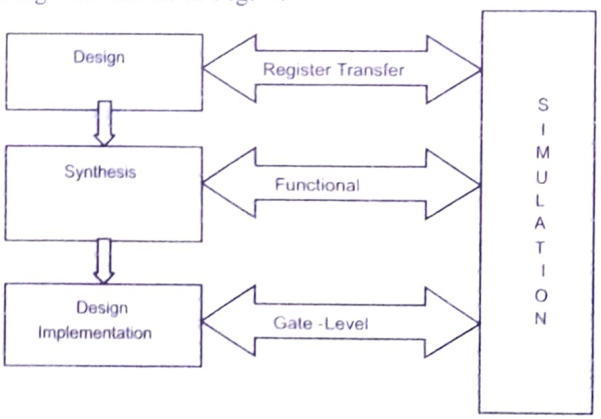

Fig. 4. Development stages of FPGA

- Design
- Simulation
- Synthesis
- Design Implementation (Place and Route + Bit Stream Generation)

2.1 Design

The design process involves conversion of requirements into a format that represents the desired digital function(s). Common design formats are schematic capture, hardware description language (HDL), or a combination of the two. Each method has its advantages and disadvantages but HDLs generally offer the greatest design flexibility.

2.1.1 Schematic capture: Schematic capture is a graphical depiction of a digital design and shows the actual interconnection between each logic gate that produces the desired output function(s). Many of these logic gate symbols involve proprietary information that is available to the designer only through the specific vendor's component library. It makes the design vendor dependent. Examples of schematic capture tools are Viewlogic's *View- Draw* and HDL's *EASE*. The main advantage of schematic capture is that the graphical representation is easy to understand. But its major drawback is an increase in cost and time to reproduce a design for different vendors due to the design's proprietary nature.

2.1.2 HDL method: Hardware description languages (HDLs) use code to represent digital functions. "Firmware" often refers to the resulting HDL code. HDLs are a common and popular approach to FPGA design. One can create the source code with any text editor. Special HDL editors like *CodeWright* and *Scriptum* (a free HDL text editor by HDL Works) offers features such as HDL templates and highlighting reserved words not found in ordinary text editors. HDLs can be generic (supported by multiple simulation and synthesis tool sets) like *Verilog* or *VHDL* (Very High Speed IC HDL), or vendor specific like Altera's Hardware Description Language (AHDL), which is only recognizable by Altera's design tool set. There are two writing styles for HDL designs: structural or behavioral.

Structural firmware is the software equivalent of a schematic capture design. Like schematic capture, a structural design uses vendor specific components to construct the desired digital functions. This type of HLD firmware is again vendor dependent like its graphical counterpart and has the same disadvantages.

Behavioral HDL firmware describes digital functions in generic or abstract terms that are generally vendor independent. This provides enough flexibility for code reuse in different vendor's FPGAs with little or no code modification. Behavioral designs have advantages of its flexibility and time and cost-savings, and it offers little to no vendor dependence. Only those components are required to be changed for designs that require vendor specific resources, such as RAM. VHDL and Verilog are the most popular HDL languages. VHDL files consist of three main parts:

- Library declaration
- Entity declaration, and
- Architecture section

An optional heading section, containing pertinent information, such as the designer's name, filename, a brief summary of the code, and a revision history, which otherwise is not required for VHDL should also be included.

i) **Library declaration** - The library declaration is the first section in the source file. This is where one places the library and package call-out statements. Libraries and packages define and store components, define signal types, functions, procedures, and so forth. Packages and libraries are standardized, such as the IEEE library, and also defined by a user (designer) or vendor. Once all the libraries and packages are visible, this section is complete.

ii) **Entity declaration** - The entity declaration section immediately follows the library declaration. Each entity has an assigned name. This section makes the I/Os visible to other source files and the design and can represent the I/Os as physical FPGA pins. VHDL designs can contain one source file or a hierarchy of multiple files. Hierarchical file structures consist of several files connected through the signals declared in their entities.

iii) **Architecture section** - The architecture section is the body of the VHDL source code and contains the circuit description. The libraries, packages, and signals work together to develop the desired functions. The format for declaring the architecture is the reserved word Architecture followed by its name. Moreover, signals not defined in the entity section are defined in this section.

The reserved word *Begin* signifies the start of the next subsection, which combines the concurrent and sequential statements. Concurrent statements update or change value at anytime. The architecture section closes by using the reserved word *End* followed by the architecture's name.

2.2 Simulation

Once the design is complete, there are two options available: a) simulate and then synthesize b) synthesize

and then simulate. There isn't a hard and fast rule stating that one must simulate before synthesis. There are advantages to each option, and designers must determine which step is most beneficial. In fact, there may be times when a designer decided to simulate following the completion of the initial design while another time decide to synthesize. Each option lets the designer detect and correct different types of errors. Simulating the design prior to synthesis allows logic errors and design flaws to be resolved early in the development process. Synthesizing lets the designer resolve synthesis errors prior to logic errors and design flaws. Ideally, the designer would perform minimal simulation, leaving the more stringent testing to a code tester.

Simulation is an act of verifying the HDL or graphical digital designs prior to actual hardware validation. The circuit's input-signal characteristics are described in HDL or in graphical terms that are then applied to the design. This lets the code tester observe the outputs' behavior. It may be necessary to modify the source code during simulation to resolve any discrepancies, bugs, or errors. Simulation inputs or stimulus are inputs that mimic realistic circuit I/Os. Stimulus forces the circuit to operate under various conditions and states. The greatest benefit of stimulus is the ability to apply a wide range of both valid and invalid input signal characteristics, test circuit limits, vary signal parameters (such as pulse width and frequency), and observe output behavior without damaging hardware. Stimulus can be applied to the design in either HDL or graphical/waveform format. Generally, it is referred to applying stimulus to the design in the form of HDL.

Some popular simulators are *Mentor Graphics' ModelSim, Aldec's Riviera, and Altera's Quantus II.*

There are three levels of simulation:

- Register transfer level (RTL)
- Functional, and
- Gate level

Each occurs at a specific place in the development process.

The initial simulation performed immediately after the design stage is the **RTL simulation** and it only verifies that the logic is correct. No realistic timing information is available to the simulator. The only timing information that can be available to the simulator is tester generated. Much like input stimulus, a tester can insert simulated or injected delays into the original HDL design. Applying test stimulus to the synthesized or optimized netlist produced by a synthesis tool is a **functional simulation**. Optimized netlists produced by non-vendors apply estimated delays that produce more realistic simulation output results. The main benefit of performing functional simulation is that it lets the tester verify that the synthesis process hasn't changed the design. Many, but not all, third-party simulation tools accept post-synthesis netlists. **Gate-level simulation** involves applying stimulus to the netlist created by the implementation process. All internal timing delays are included in this netlist, which provides the tester with the most accurate design output. Many third-party simulation tools can perform gate simulation. Ideally, each level of simulation is performed at the appropriate development stage.

Each simulation level offers various benefits. RTL uncovers logic errors, the functional level verifies that the pre- and post-synthesis designs are equivalent, and the gate level uncovers timing errors. Opting to omit simulation and testbenching will generally cost the project additional time and money. Simulation is valuable and as a guideline, at least 2 times the number of hours spent writing the code should be spent developing and testing the code.

2.3 Synthesis

Synthesis is the process that reduces and optimizes the HDL or graphical design logic. Some third-party synthesis tools are available as a part of the FPGA vendor's complete development package. Synplicity's Synplify and Mentor Graphics' Leonardo Spectrum, Precision RTL, and Precision Physical are examples of third-party synthesis tools. Xilinx offers ISE Project Foundation, which is a complete development application that includes a synthesis tool. Altera has Quartus II Integrated Synthesis, QIS.

Although some FPGA vendors offer synthesis, they still recommend using a third-party's synthesis tools. The synthesis tool must be set up prior to actually synthesizing the design. FPGA information includes the vendor's name, the specific part or family, the package type, and the speed. The synthesis process takes this information and the user-defined constraints and produces the output netlist. A constraints file specifies information like the critical signal paths and clock speeds. After completing set-up, synthesis can begin. General synthesis flow for tools like Synplicity's Synplify involves three steps, creating structural element, optimizing, and mapping. Figure 5 shows a synthesis flow diagram.

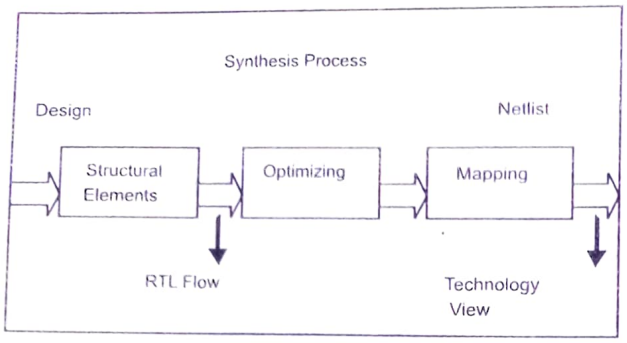

Fig. 5. Design synthesis flow diagram

The first step in the synthesis process is to take the HDL design and compile it into structural elements followed by optimizing the design that makes it smaller and faster by removing unnecessary logic and allowing signals to arrive at the inputs or output faster. The goal of the optimizing process is to make the design perform better without changing the circuit's functions. The final step involves mapping or associating the design to the vendor specific architecture. The mapping process takes the design and maps or connects it using the architecture of the specific vendor. This means that the design connects to vendor specific components such as look-up tables and registers. The optimized netlist is the output of the synthesis process. This netlist may be produced in one of several formats. *Edif* is a general netlist format accepted by most implementation tools, while .xnf format is specific to Xilinx and is only recognized by Xilinx's implementation. In addition to the optimized netlist, many synthesis tools like Synplify will produce a netlist for gate-level simulation and other report files. Stimulus applied to this netlist instead of the original HDL design produces the functional- level simulation, which lets the designer verify that the synthesis process hasn't changed the design's functions.

2.4 Design implementation (Place and Route+ Bit Stream Generation)

The final stage in the FPGA development process is the design implementation, also known as **place and route (PAR)**. Each FPGA vendor has its own implementation tool, such as Xilinx's has *Project Navigator* and Altera's *has Quartus II's*. If the FPGA vendor has a complete development tool, meaning it can perform synthesis, and the design is synthesized using this tool, little or no setup is required for PAR. However, if a third-party synthesis tool is used, the implementation tool must be set up, which involves directing the PAR tool to the synthesized netlist and possibly a constraint file. The constraint file contains information such as maximum or minimum timing delays for selected signal(s) and I/O pin assignments. Pin assignments can be automatic (performed by the tool) or manual (dictated by the designer). Automatic pin assignment is generally the best option for new designs, as it lets the tool more effectively route the design without having fixed pin assignments. It may be necessary to manually assign signals to specific pins to achieve easy board routing, to provide the minimum signal route for timing-critical signals, or be compatible with legacy designs. There are numerous reasons why manual pin assignments would be necessary.

But regardless of the reason, the designer must make this information available to the PAR tool, which is done by creating a user constraint file that's used by the PAR tool. After completing setup, the PAR process can begin. Xilinx's Foundation or Project Navigator performs design implementation in three steps: translate, fit, and generate programming file. Translate, involves verifying that the synthesized net list is consistent with the selected FPGA architecture and there are no inconsistencies in the constraint file. Inconsistencies would consist of assigning two different signals to the same pin, assigning a pin to a power or ground pin, or trying to assign a non-existing design signal to a pin. In such cases the translate step will fail and the implementation process will be stopped. Translate errors must be corrected before advancing to next step of fit stage. This step involves taking the constraints file and netlist and distributing the design logic in the selected FPGA. If the design is too large or requires more resources or available logic than the selected device offers, the fitter will fail and halt the implementation process. To correct this type of error, the current FPGA is replaced with a larger one and is re synthesized, and PAR is repeated for the design. A successful fit stage is necessary to proceed to generate the programming file stage. The final step is to generate the programming file, which can be stored in flash memory, PROMs, or directly programming into the FPGA. This process is also called **Bit Stream Generation**. Joint Test Action Group (JTAG) and third-party programmers like Data I/O are the two programming methods that are used to store the programming file in memory. The appropriate format depends on the FPGA vendor, the programming method and the device used to hold the programming. In addition to the implementation process creating the programming file, there are several output report files created, such as a pad file which contains information such as signal pin assignment, part number, and part speed.

This is how the FPGA designer completes his design process following the four development stages.

3. ISSUES AND CHALLENGES IN FPGA BASED DIGITAL DESIGN

Field Programmable Gate Array (FPGA) have become an attractive implementation solution in the modern digital systems due to its reconfigurable architecture, ease of design and flexibility, better performance and low non recurring engineering cost (NRE). However, following two are significant factors and issues of concern in FPGA design:

- Power Consumption
- Radiation Effects on FPGAs

3.1. Power Consumption: Programmability of FPGAs result in with more loaded interconnection network as compared to customized circuits. As a result, pass transistors, signal buffers and other programmable switching structure increase the capacity load of signal networks over dedicated metal wires. Therefore, there is a significant increase in the power consumption on account of flexibility of FPGA as compared to other processing/ implementation units having fixed architectures and interconnections.

There are two primarily types of power consumptions in FPGAs: static and dynamic[2],[7],[13],[14]. Static power is consumed due to transistor leakage whereas dynamic power is consumed by toggling nodes and is mainly a function of voltage, frequency, and effective capacitance. It is important to understand both types of variations under various conditions so that they can be properly optimized to meet the design's power budget.

3.1.1 Static Power: Static power is mainly caused by leakage current between power supply and ground and consists of sub threshold leakage, reverse biased PN junction current, gate induced drain leakage and gate tunneling[8]. The leakage current starts to be fairly significant at 90 nm for both ASICs and FPGAs and becomes even more challenging at 65 nm. Transistor leakage and hence static power varies with the following parameters: Process, Voltage and Temperature. It has been revealed that static power increases dramatically along with shrinking transistor size. Moreover, the thermal characteristics also get affected significantly with design shrinking of features. The threshold voltage of the transistor that also increases the leakage needs to be lowered to obtain higher performance from the transistor. Static power and leakage are also influenced by core voltage and variation is approximately square and cube of core voltage. Static power increases approximately 15% with increase of only 5% of core voltage. Leakage is strongly influenced by the junction or die temperature[20]. On-chip temperature of processing unit may vary among the whole area. The maximum on-chip area is related to the chip area and maximum power dissipation. The reduction of total maximum power therefore may increase the maximum temperature that further influences the static power.

To reduce the transistor leakage FPGAs, Xilinx IC designer started to adopt the use of a third-gate oxide thickness (triple oxide) in the transistors of 90 nm Virtex-4 FPGAs. The third medium thickness oxide (midox) and higher threshold voltage in the portion of the transistors of Virtex-4 FPGAs allows a dramatic reduction in overall leakage compared to other FPGAs. Virtex-5 FPGAs continue to deploy the triple oxide technology in the 65 nm process node that enables significant lower leakage current and static power.

3.1.2 Dynamic Power: Dynamic power is the power-consumed during switching events in the core or I/Os of an FPGA. This is caused by signal transition at device transistors and frequency of signal transition is obviously related to clock frequency. In FPGA, the dynamic power consumption is design dependent due to its programmability. The factors like switching activity of resources, effective capacitance of resource and resource utilization are design dependent and contribute to dynamic power. Switching activity represents the average number of signal transition in a clock cycle. The effective capacitance corresponds to the sum of parasitic effect due to interconnection wires and transistors. FPGA architecture usually offers more resources than actually required to implement a particular design, which means some resources, are not used after the final chip configuration and they don't consume dynamic power; this is referred to as resource utilization. Taking all these factors into account, total dynamic power consumption of the device is generally modeled as:

$$P = V^2 f \sum_n S_n C_n U_n$$

Where:

n is the number of toggling nodes,

V is supply voltage,

f is clock/toggle frequency (presumed to be fixed for each resource) and

S, C, and U correspond to switching activity of resources, effective capacitance of resource and resource utilization respectively[7],[8],[14].

All nodes in the FPGA consume power through a combination of charging transistor parasitic capacitance

and metal interconnect capacitance. The later depends on the length of route in FPGA, while the number of transistors that are switching determines the node capacitance. Reducing the number of switching transistors and minimizing routing lengths through tighter packing can reduce the dynamic power. The Virtex-5 FPGAs have lowered the gate capacitance and shorter interconnect traces that contributes to lowering the node capacitance by about 15% and hence lowers the dynamic power. Moreover dynamic power gets reduced by approximately 17% in Virtex-5 FPGAs simply by decreasing core voltage from 1.2 V to 1.0 V. Table 1.1 shows the relative dynamic power savings in Virtex-4 and Virtex-5 FPGAs as a result of reduction of capacitance and core voltage. Process and temperature also cause little variation of 5-10% in dynamic power[20].

Table 1.1: Relative Dynamic Power Saving in Virtex-4 and Virtex-5 FPGAs

Parameter	Virtex-4 FPGA 90 nm	Virtex-5 FPGA 65 nm	% of change
Core Voltage	1.2	1.0	-16.6%
Total Capacitance	1.0	0.85	-15%
Dynamic Power	1.44	0.85	-40%

In the recent past, research efforts in reducing the power consumption and improving the power efficiency of reconfigurable FPGAS have intensified. Three major possible strategies for reduction of power consumption in FPGA are:

- By simplifying the algorithm used at system level
- By changing logic partitioning, mapping placement and routing at designer level, if architecture is fixed and
- By enhancing the operating conditions of devices including power supply, clock frequency and capacitance.

Some of the techniques were proposed for reducing leakage power by disabling unused portion of the FPGA and by selecting polarities for logic signals at the input of LUTs so that they spend the majority of their time in the low leakage states. The proposal of application of voltage scaling onto the logic blocks of FPGA architectures was followed by the technology mapping technique to utilize this feature more efficiently. Studies were also extended to create programmable dual Vdd architecture to operate certain blocks at either high Vdd or low Vdd[11]. Gayasen proposed to apply dual supply voltage on logic blocks and routing multiplexers[12]. Scaling down the supply voltage is a popular design technique and has been successful in ASICs for with dual Vdd or multi Vdd designs for power reduction[1],[5],[10]. Mondal proposed dual Vdd dual Vt routing architecture, where a fraction of routing tracks operate on high Vdd high Vt level while rest of the routing tracks operate on scaled down Vdd and Vt[15]. Dual voltage supplies were provided to driving buffers of the routing segments as well as switch matrix. This architecture can also be used along with the earlier above-mentioned power reduction techniques.

3.2 Radiation Effects

Increased density and corresponding shrinkage of process geometry has made FPGA devices more susceptible to failures due to external radiations. Earlier this has been an issue for space based system but is now becoming an issue for terrestrial systems in elevated radiation environment and commercial avionics as well. Radiation effects on single FPGA have system level consequences and will need to be addressed in current and future designs. There are two main categories of radiation effects that are relevant for Static Random Access Memory (SRAM) Field-Programmable Gate Arrays (FPGAs) in space: Total- Dose Effects and Single-Event Effects (SEEs).

Total-Dose Effects are cumulative effects that induce degradation of electrical parameters at the device, circuit, and system levels. They are induced by the total amount of ionizing energy deposited by photons or particles such as electrons, protons, or heavy ions. This effect is similar to sunburn to human and is dependant on the amount radiations and how long it took to accumulate the total dose.

SEEs are induced by the passage of a single high energy proton or heavy ion through a device or a sensitive region of a microcircuit. SEEs can be either destructive (e.g., Single-Event Latch-up [SEL]), or non-destructive, such as the occurrence of transient faults in combinational and sequential logic.

The main reliability issues in radiations environment are: Single Event Latch up (SEL), Single-Event Upset (SEU) and Multiple-Bit Upset (MBU)

Single event latch up (SEL) occurs when one of the parasitic bipolar transistor created as a by product of CMOS fabrication process is activated by a charged

particle. This type of upset activated by a charged particle is very serious and results in a short being created from power to ground on the chip. However, special fabrication processes using epitaxial substrate eliminate the parasitic bipolar transistors and susceptibility to SEL. Moreover, increased density of newer device families and the corresponding lower core voltage is making SEL less likely.

Single events upsets (SEUs) occur when RAM cell's state gets changed due to exposure to energetic particles. The effete will be determined by the function of particular RAM and it can alter one or more of the following:

- logic content
- user logic state
- logic configuration
- routing

Altered logic content is the most straight forward effect and results in a flip flop transitioning to incorrect state.

Altered user logic results in a glitch or momentary bad data if it is not a part of feedback element and goes unnoticed. But this error will be persistent and very likely to cause undesirable operations if it is a part of feedback element. FPGAs support global functions for programming, initialization and debug. These functions get improperly activated with altered logic content due to SEU and cause the device to reset or enter configuration mode and interrupts all user functionality immediately. These events require complete reconfiguration for recovery and are known as single event functional interrupts (SEFIs).

Altered logic configuration bits are always persistent where the logic function gets changed. These errors are detectable via configuration memory readback and are easily repairable via partial reconfiguration. However, the user logic will likely malfunction randomly during the time the logic is altered.

Altered routing is least likely to cause a logic failure but is statistically most likely effect of SEU. There is high probability that the upset will connect the unused wires and will be don't care as far as logic is concerned. But as parasitic segments are added to the design due to routing faults, there is a gradual rise in the device power consumption. Moreover, it will have the effect of degrading the timing margin and finally may cause logic failures if not repaired by partial reconfiguration. The shorts and open caused due to altered routing in the wires utilized for design, of course, will have immediate persistent effect.

Multiple-Bit Upset (MBU)

Multiple Bit upset is an SEU that results in more than one adjacent bits flipping due to an oblique angle strike. Its probability steadily increases as geometries shrink. Use of maximum MBU distance observed is useful to determine block RAM interleaving required so that even MBUs are able to be corrected by the Error Correcting Code (ECC)

3.2.1 Mitigation Techniques

Using some redundancy techniques targeting the Field-Programmable Gate Array (FPGA) architecture can protect the design at the high-level description VHDL or Verilog level. The most popular high-level Single-Event Upset (SEU) mitigation technique currently used to protect designs synthesized in the SRAM-based FPGAs is **triple modular redundancy (TMR) combined with scrubbing**. Xilinx has released the tool called X-TMR that automatically implements TMR into the user description[18]. The user himself can also implement TMR in his design that provides immunity from a single configuration or state upset[6]. However, due to the high area overhead of TMR, some alternative solutions have been proposed in recent years. Therefore, the user has the flexibility of implementing duplication and self-checking techniques instead of TMR. These techniques may compromise the fault tolerance at some point, but the final result may be acceptable for a set of applications. In this way, it is possible to use a commercial FPGA part to implement the design and the soft error mitigation technique is applied to the design description before being synthesized in the FPGA. The user has the flexibility of choosing the fault-tolerant technique and consequently the overheads in terms of area, performance, and power dissipation. One very important step of the design flow is the validation of the fault tolerance technique, which is usually done by fault injection. A circuit or a tool in the computer can modify the original bit stream configured into the FPGA by flipping the bit stream bits, one at a time. This flip emulates a SEU in the configuration memory cells. The output of the design under test (DUT) can be constantly monitored to analyze the effect of the injected fault into the design. If an error is detected, this means that the fault-tolerant technique implemented is not robust for that specific fault (SEU) in that target-configuration memory bit. Table 1.2 presents a summary of Single-Event Effect (SEE) issues and possible SEU mitigation solutions[16].

faults would eventually break the redundancy. It is recommended to scrub at least 10X faster than the worst-case SEU rate. When the FPGA is in this mode, an external oscillator generates the configuration clock that drives the FPGA and Programmable Read-Only Memory (PROM) that contains the "golden" bit stream. At each clock cycle, new data are available on the PROM data pins. The frequency that scrubbing must be performed depends on the particle flux and cross-section of the device. Xilinx Vertex devices support readback and configuration mode that operates on only a portion of the device and is known as partial readback and configuration[3]. It allows a more efficient means of repairing configuration upsets. Unlike complete configuration, it does not reset the device and allows the uniterrrupted operation of the user.

c) **TMR Tool**

Implementing TMR is very difficult if it is done manually. A special software tool (TMRTool) has been developed and fits within the Xilinx design flow. This tool eliminates half-latches (weak keepers), which are also sensitive to SEU. This tool has been evaluated in several radiation tests, but more efforts will be required to ensure that it is completely effective.

TMR does not come without a price. Obviously, designs are at least 3 times as large as a non-TMR design, and suffer from speed degradation as well. In particular, feedback TMR degrades the speed of operation by introducing a longer feedback path. Power consumption is also tripled along with the logic. The underlying assumption of TMR is that only one upset will occur within a given logic block. This is not always a good assumption to make. In Virtex II devices, recent testing resulted in approximately .3-.5% of upsets causing multiple bit upsets within the device. Also, the scrubbing frequency defines the rate at which upsets can be detected – this combined with the rate of upsets provides the actual tolerance of the design[6]. This being said, a proper TMR implementation combined with fast scrubbing can provide better than an order of magnitude increase in the radiation tolerance of a given design.

SRAM based FPGAs are widely used due to their density, cost, and in system programmability. However, another option exists in antifuse technology. In addition, antifuse vendors also offer rad-tolerant versions of some product lines, which are intrinsically resistant to SEUs to a degree not available in SRAM devices. Antifuse has several advantages to SRAM. These one time programmable devices use physical shorts between metal routing layers to configure their logic. Aside from being faster and more power efficient than comparable SRAM based switches, they are immune to radiation effects. As can be seen from table 1, this eliminates 97% of sensitive bits (in a device of similar density). Application of TMR in an antifuse part is usually less costly in resources, as only the state dependent logic needs to be triplicated. The more efficient logic switching results in lower power consumption and quieter operation that are important considerations in mixed mode designs. The main drawback of antifuse is its one time programmability; it is best suited for applications where the initial requirements are stable and not expected to evolve over time. In addition, antifuse parts are not available in as high logic densities as SRAM devices.

Some antifuse vendors provide rad-hard versions of some of their product lines. These devices are even more radiation tolerant than standard antifuse, with internal flip-flops TMRed in silicon (a device by Quicklogic/Aeroflex even has hardware TMRed RAM arrays). These devices completely remove the need to TMR user designs, and are suitable for the highest reliability requirements. However, the selection of devices is constrained, and is not available in the highest densities supported by antifuse.

4. CONCLUSION

Modern FPGAs are already at the heart of most low to mid volume electronic systems, and their capabilities will continue to improve in the future. The process technology trends in FPGA manufacturing indicate that the leakage power will be an increasing important design concern for future reconfigurable devices. Moreover, with the continuous shrinking of device geometry, the susceptibility to radiation upset will continue to grow. Upset tolerant design techniques, both from a system and device level, are already becoming a requirement for many systems. SRAM FPGAs, such as the Xilinx vertex series, have long been favoured due to their unmatched performance, density, and in system programmability, yielding a powerful and flexible solution chosen by many designers. However, their relatively high susceptibility to radiation upset is a factor to be considered in a growing number of environments.

REFERENCES

[1] K. Usami and M. Horowitz, "Clustered voltage scaling techniques for low-power design," *ISPLED*, 1995.

[2] G.J.M. Smit and P.J.M. Havinga, "A survey of energy saving techniques for mobile computers", internal report University of Twente, 1997.

[3] Virtex architecture guide, Xilinx San Jose CA 9/00.

[4] Carmichael C., Caffrey M., and Salazar A., XAPP216, "Correcting Single Event Upsets through Virtex Partial Configuration," June 2000.

[5] M. Hamada, Y. Ootaguro, and T. Kuroda, "Utilizing surplus timing for power reduction," *Proc. CICC*, 2001.

[6] Triple modular redundancy design technique for Virtex FPGAs (xAPPP197, v 1.0), C, Carmichael, Xilinx, 11/01.

[7] L. Shang, A. S. Kaviani, and K. Bathala, "Dynamic Power Consumption in Virtex-II FPGA Family", *Proceedings of the 2002 ACM/SIGDA 10th International Symposium on Field Programmable Gate Arrays*, pages 157 – 164. ACM Press, 2002.

[8] H.G. Lee, S. Nam, and N. Chang, "Cycle-accurate energy measurement and high-level energy characterization of FPGAs", *Quality Electronic Design, 2003. Proceedings. Fourth International Symposium* on 24-26 March 2003, Page(s): 267 – 272.

[9] O.S. Unsal and I. Koren, "System-level power-aware design techniques in real-time systems", *Proceedings of the IEEE, Volume 91, Issue 7*, July 2003 Page(s):1055 – 1069.

[10] R. Puri, L. Stok, J. Cohn, D. Kung, D. Pan, D. Sylvester, A. Srivastava, and S. Kulkarni, "Pushing ASIC Performance in Power Envelope," presented at Design Automation Conference, 2003.

[11] F. Li, Y. Lin, and L. He, "FPGA Power Reduction Using Configurable Dual-Vdd," presented at Design Automation Conference, San Diego, CA, 2004.

[12] A.Gayasen, K.Lee, N.Vijaykrishnan, M.Kandemir, M.J.Irwin, T.Tuan, "A Dual-Vdd Low Power FPGA Architecture", *International conference on Field Programmable Logic and Its Application, 2004, Antwerp, Belgium.*

[13] Steven J. E. Wilton, Su-Shin Ang, and Wayne Luk, "The Impact of Pipelining on Energy per Operation in Field-Programmable Gate Arrays", *Field Programmable Logic and Application: 14th International Conference, FPL 2004*, Leuven, Belgium, August 30 September 1, 2004. *Proceedings. Volume 3203 / 2004*, Chapter: pp. 719 – 728.

[14] V. Degalahal and T. Tuan, "Methodology for high level estimation of FPGA power consumption", *Design Automation Conference, 2005. Proceedings of the ASP-DAC 2005. Asia and South Pacific, Volume 1*, 18-21 Jan. 2005 Page(s):657 – 660 Vol. 1.

[15] Mondal, Memik, A low power FPGA Routing Architecture, ECE Department Northwestern University, Evanston IL USA, 2006.

[16] Label K., M. Berg, D. Black, W. Robinson, and A. Jordan, "Trade Space Involved with Single Event Upset (SEU) and Transient (SET) Handling of Field Programmable Gate Array (FPGA) Based Systems," 2006 Workshop on Hardened Electronics and Radiation Technology (Heart), 2006.

[17] Carmichael C. XAPP197, "Triple Modular Redundancy Design Techniques for Virtex FPGAs," June 2006.

[18] TMR Tool user guide, Version 6.2.3, Xilinx Inc., Sept. 2006.

[19] Pawel P. Czapski and Andrzej Sluzek, Power Optimization Techniques in FPGA Devices: A combination of system and low level, World Academy of Science, Engineering and Technology May 10, 2007.

[20] Peggy Abusaidi, Matt Klien and Brain Philofsky Virtex-5 FPGA, system power design considerations WP2859(v1.0) Feb 14, 2008.