Circuits and Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ens.ewi.tudelft.nl/

CAS-2010-06

M.Sc. Thesis

A Power-Aware Fault-Tolerant Hardware System for a Custom Reconfigurable Platform

Kotaro Kobayashi

Abstract

Electronics systems in deep-submicron era face many new challenges. Increased intricacy of the manufacturing process will likely to increase the manufacturing defect while testing of those effect will be very challenging. Smaller feature size will also face new reliability issues due to phenomenas such as Joule heating and electromigration. Furthermore, chances of temporal defects, namely soft-errors, or Single Event Fault (SEU) will increase because the critical charge, the charge required to flip a logical value in flip-flops decreases with technology scaling.

Power consumption is another issue that is ever greater in electronics design. Technical, financial, and ecological concern all require devices that consumes as small amount of energy as possible.

The Ubichip, a bio-inspired reconfigurable VLSI developed in PERPLEXUS European project has attributes such as dynamic self replication and dynamic routing capability, both of which may help develop a system to increase reliability while addressing the power consumption issues.

The author has designed a power aware fault tolerant system based on the Triple Modular Redundancy (TMR) fault tolerant strategy to be implemented on Ubichip. The system also controls the number of functional unit to regulate the overall system power consumption. This report describes the design, implementation, and simulation of the fault-tolerant system.

TU Delft

Faculty of Electrical Engineering, Mathematics and Computer Science Delft University of Technology

A Power-Aware Fault-Tolerant Hardware System for a Custom Reconfigurable Platform

A Built-In-Self-Repair Circuit Implementation on Ubichip

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Electrical Engineering - Microelectronics

by

Kotaro Kobayashi
born in Tokyo, Japan

This work was performed in:

Circuits and Systems Group
Department of Microelectronics & Computer Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft University of Technology

Delft University of Technology
Department of
Microelectronics & Computer Engineering

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"A Power-Aware Fault-Tolerant Hardware System for a Custom Reconfigurable Platform"** by **Kotaro Kobayashi** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: October 2010

Chairman: _____

Prof.dr.ir. Alle-Jan van der Veen

Advisor: _____

Dr.ir. T.G.R.M. van Leuken

Committee Members: _____

Dr.ir. S. Cotofana

_____

Abstract

Electronics systems in deep-submicron era face many new challenges. Increased intricacy of the manufacturing process will likely to increase the manufacturing defect while testing of those effect will be very challenging. Smaller feature size will also face new reliability issues due to phenomenas such as Joule heating and electromigration. Furthermore, chances of temporal defects, namely soft-errors, or Single Event Fault (SEU) will increase because the critical charge, the charge required to flip a logical value in flip-flops decreases with technology scaling.

Power consumption is another issue that is ever greater in electronics design. Technical, financial, and ecological concern all require devices that consumes as small amount of energy as possible.

The Ubichip, a bio-inspired reconfigurable VLSI developed in PERPLEXUS European project has attributes such as dynamic self replication and dynamic routing capability, both of which may help develop a system to increase reliability while addressing the power consumption issues.

The author has designed a power aware fault tolerant system based on the Triple Modular Redundancy (TMR) fault tolerant strategy to be implemented on Ubichip. The system also controls the number of functional unit to regulate the overall system power consumption. This report describes the design, implementation, and simulation of the fault-tolerant system.

Acknowledgments

This work has been partially funded by the European Union (PERPLEXUS project, Contract no. 34632).

I would like to thank my advisor Dr.ir. T.G.R.M. van Leuken at TU Delft, as well as Dr. Juan Manuel Moreno Arostegui at Universitat Politecnica de Catalunya (UPC) for their assistance during the writing of this thesis. Without them, this would not have been possible.

Kotaro Kobayashi
Delft, The Netherlands
October 2010

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

The IC technology scaling, which follows the famous Moore's law has evoked a great deal of advancement in modern electronics for the last few decades. Designers have been able to integrate greater number of transistors on a limited area of silicon die; modern VLSI systems with multiple function blocks on a single die allow designers to reduce the physical size of the systems and manufacturing costs. The ITRS predicts in [8] that the gate length of VLSI systems will go below 20 nm in the later half of this decade, a length enough to fit only few hundreds of silicon atoms in one line. This deep-submicron paradigm poses new challenges to the VLSI design; intricacy of the fabrication will be greater, so that manufacturing defects will likely to increase while testing for those defects will be very challenging due to the ever increasing complexity of the system. The reliability will also suffer due to phenomena such as gate insulator tunneling, Joule heating, and electromigration. Furthermore, the small feature size will certainly increase the unpredictable errors due to alpha particles, namely soft error, or Single Event Upset (SEU) [4], [8].

There have been many advancements in techniques such as Design for Test (DFT) and Built-in Self-test (BIST) [4]. While these tests can effectively detect faults due to defects, they cannot detect unforeseeable faults caused by aging-defects or temporal faults such as SEU. In order to assure the reliability while incorporating deep-submicron technologies, the system should have dynamic fault-tolerance capabilities to detect and correct errors on the run. If a VLSI system can autonomously detect and correct an error situation dynamically, it will not only increase the reliability but also the yield and life-time of the ICs, resulting in a significant cost reduction [17].

The Ubichip is a bio-inspired custom reconfigurable VLSI system developed in the PERPLEXUS project [22], [31]. Ubichip offers bio-inspired capabilities such as dynamic routing, self-replication, and neural networking. The operational flexibility provided by these mechanisms gives Ubichip a great potential for implementing dynamic fault tolerant systems with Built-in Self Repair (BISR) capabilities.

1.2 Thesis Goals

The goal of this project is to design and implement a proof-of-concept design of a power aware fault tolerant system on the Ubichip bio-inspired hardware platform. Successful result can confirm that a bio-inspired hardware platform is not only capable

of implementing fault tolerant system efficiently but also capable of reacting to the system power consumption by regulating the area of active logic circuit so as not to exceed the defined power consumption limit.

1.3 Results

- A power aware fault tolerant system, which has Triple Modular Redundancy (TMR) for the fault masking and Built-in Self-Repair (BISR) for the fault repair was designed
- The design was successfully implemented and simulation confirmed the correct operation as the actual fabricated chip was not available
- A report of the design and implementation was written and accepted for 2010 International Conference of Evolvable Systems (ICES) [10]
- Issues of Ubichip platform for this application was identified
- Based on the experience, recommendations for the future research were listed
- The design was implemented and functions were verified in standard VHDL environment, which realized some functions that were not possible on Ubichip environment

1.4 Outline

This report is organized as follows:

Chapter 2 explains the background study conducted prior to the implementation, underlining information for the motivation as well as the relevant information on Ubichip reconfigurable platform is explained.

Chapter 3 explains the detailed architectural design of the implementation including the overall design and the system requirements.

Chapter 4 explains implementation and functional test of the design on Ubichip platform.

Chapter 5 explains the implementation of the design on standard VHDL environment. This implementation complements the Ubichip experiment by realizing functions that was not possible on Ubichip.

Finally, Chapter 6 contains the Analysis of the implementation, list of the identified issues, recommendations for the future research, and the concluding remarks.

2 Background Study

2.1 Motivation: Issues in Modern Reconfigurable Circuits

This project, an implementation of power aware fault tolerant system addresses two of the major issues facing electronics design today: reliability and power consumption, which are particularly serious in reconfigurable hardware. Ubichip, a custom bio-inspired reconfigurable platform has attributes that can be useful in mitigating those issues.

2.1.1 Reconfigurable Devices

More and more electronics systems have been incorporating Field Programmable Gate Array (FPGA) in their designs in last decades. Programmable gate arrays, which used to be used mainly for prototyping and debugging in digital system design phase, are now small, reliable, and inexpensive enough to be used for final hardware as well. Use of such devices brings a lot of benefits to the designers: single hardware platform can be used for different applications, design phase can be significantly shortened because of simpler debugging capability and elimination of silicon design phase, design mistakes can be corrected after product is in use, design can continuously be upgraded, and modification of the design can be done remotely.

Many attributes of the reconfigurable devices make them very attractive for fields such as space application, where remote update is the only option available. Despite those clear benefits, there is a fundamental issue in reconfigurable devices that makes their use in space application challenging; SRAM based reconfigurable devices have lower tolerance to radiation environments and reconfigurable devices consume more power than custom non-reconfigurable IC of equivalent circuit. Radiation in the environment can cause serious problems to LSI such as Single Event Upset (SEU), Latch-up, and devise deterioration.

2.1.2 Reliability Issues

Problems the radiation can cause on electronics systems have been known for decades. Back in 1980s the problems were mostly limited to applications for high radiation environment such as space, high altitude, nuclear, and military applications. Today, radiation induced problems still remain persistent in those applications but the issue has also become serious concern for electronics systems for application field other than those in radiation intensive environments due mainly to technology scaling in

Integrated Circuit.

Types of degradation and failures caused by radiation varies. Those radiation include photons (X-rays and gamma rays), Charged particles and cosmic rays (Electrons, protons, alpha particles, heavy ions), and uncharged particles (neutrons). Problems range from permanent degradations such as collapse of depletion region and characteristic change of the transistors to temporary failures such as latchup and data upset [24].

The major radiation caused problem is the soft error. Also called Single Event Upset (SEU) or transient fault, values of the electronic signals are altered to cause erroneous operation of the system even though there is no design or manufacturing defects. Permanent faults can only be mitigated by incorporating prevention measures at design and manufacturing stage. There are varieties of radiation hardened IC packages available for military and aerospace applications. Temporary failures on the other hand can be mitigated dynamically, which in turn is the target problem of this project.

Soft errors occur when radiation hitting on the circuit causes enough accumulation of charges at circuit nodes that they exceed the critical charge (Q_{CRIT}); making the value of the signals to alter. It is reported in [2] that if soft errors are not corrected, it induces the error rate higher than all the other failures combined. Furthermore, data storage blocks, especially SRAM circuits are far more susceptible to radiation induced SEUs because SRAM circuits do not have masking effects that may mask the fault even if a SEU occurs at one node [25].

The equation below shows the variables affecting the Soft Error Rate (SER). One can see from this equation that the critical charge (Q_{CRIT}) and charge sensitivity (Q_S) are the key parameters. Device scaling, by making the feature size smaller decreases critical charge mainly due to decreased supply voltage though the efficiency increases because smaller transistors are more sensitive to particle strikes.

$$SER \propto F \times A \times \exp\left(-\frac{Q_{CRIT}}{Q_S}\right) \tag{2.1}$$

F: Radiation flux with energy > 1 MeV in particles/$(cm^2 \cdot s)$
A: Area of the circuit sensitive to particle strikes in cm^2
Q_{CRIT}: Critical charge in fC
Q_S: Collection efficiency of the device in fC

There are three masking effect that may conceal the effect of SEU in digital circuit: Logical Masking, Electrical Masking, and Latching-window Masking. Logical maskings occur when a node with SEU at the time of occurrence is isolated from the output hence the SEU not affecting the overall functionality. Electrical masking occurs when multiple stage of logic gates attenuate the voltage change of the node due to SEU; resulting in SEU not affecting the circuit operation. Latching-window masking occurs when a SEU happens outside the latch window (setup & hold); the altered value of

the signal does not pass the latch in this case. SEU in SRAM changes the value stored in flip-flops so that none of the three masking effect can reduce the effect of SEU on SRAM circuits. As a result, SRAM based reconfigurable devices are highly susceptible to Radiation induced errors.

2.1.3 Power Issues

Technology scaling also increases the power consumption of digital circuits as discussed in [23]. Power consumption is the major concern for any electronic devices today; power consumption of systems such as large servers for data centers need to be decreased for financial and ecological reasons while power consumption directly affects the operational performance of battery driven devices.

This project involves implementation of Triple Modular Redundancy (TMR) system, which naturally increases the power consumption as the system require three copies of the identical circuit. In order to keep the system operating correctly, the power consumption must be monitored and controlled not to exceed the maximum available supply.

2.2 Fault Tolerant Systems

Fault tolerant systems are heavily used in aerospace and automotive applications, wherein safety is critical or the harsh environment to electronics systems exists. General overview of fault tolerant electronics systems is explained in[21]. Designing of System on Chip (SoC) for automotive application is explained in [1]. Bowman and his team at Intel have developed an Error Detection Sequential (EDS) circuit to address errors in microprocesors caused by manufacturing process variability [3]; an emerging major cause of faults in deep sub-micron era. For applications that are more similar to this project, designs of fault tolerant systems for space applications implemented on Xilinx FPGA platform are explained in [5] [6].

A fault in electronic system is a part that has unintended shape; fault can be a physical defect, an imperfection, or a flaw that exists in the hardware or software component. Faults cause errors, which lead to the failure of the electronics systems as represented in the figure 2.1. It is important for the designers of fault-tolerant systems to identify the types of faults they are trying to mitigate. There are various types of faults with different cause and effect; a stuck at fault would behave differently from open-circuit faults, and faults in digital circuit has different effect from faults in analog circuits. An appropriate measure needs to be taken to address a specific type of fault. Table 2.1 shows different properties of faults in electronic systems.

Fault Characteristics				
Cause	Nature	Duration	Extent	Value
- Specification Mistakes - Implementation Mistake - External Disturbances - Component Defects	- Hardware (Analog) - Hardware (Digital) - Software	- Permanent - Intermittent - Transient	- Local - Global	- Determinate - Indeterminate

Table 2.1: Fault Properties

Figure 2.1: Three Universe Model: cause-and-effect relations among faults, errors, and failures

2.2.1 Fault Tolerance Strategies

In depth study of fault tolerance strategies are discussed in literatures [9] [16] [4] [13]. In this section the basics of fault tolerant strategies relevant to the project are discussed.

The Figure 2.2 shows the relations among factors that lead to electronics system failure. A system failure is a result of error, which can either be caused by hardware faults or software faults. Software and hardware faults are result of various mistakes such as specification mistake, implementation mistake, external disturbances, and component defects. As discussed in the earlier sections, modern electronics systems that are taking advantage of scaling silicon manufacturing technologies are especially vulnerable to external disturbances and component defects. For space applications, external disturbances mainly from radiation is a serious concern.

There are seven basic fault tolerance strategies to be discussed in this section:

1. Fault Avoidance
2. Fault Masking
3. Fault Detection
4. Fault Containment
5. Fault Diagnosis
6. Repair/Reconfiguration
7. Recovery

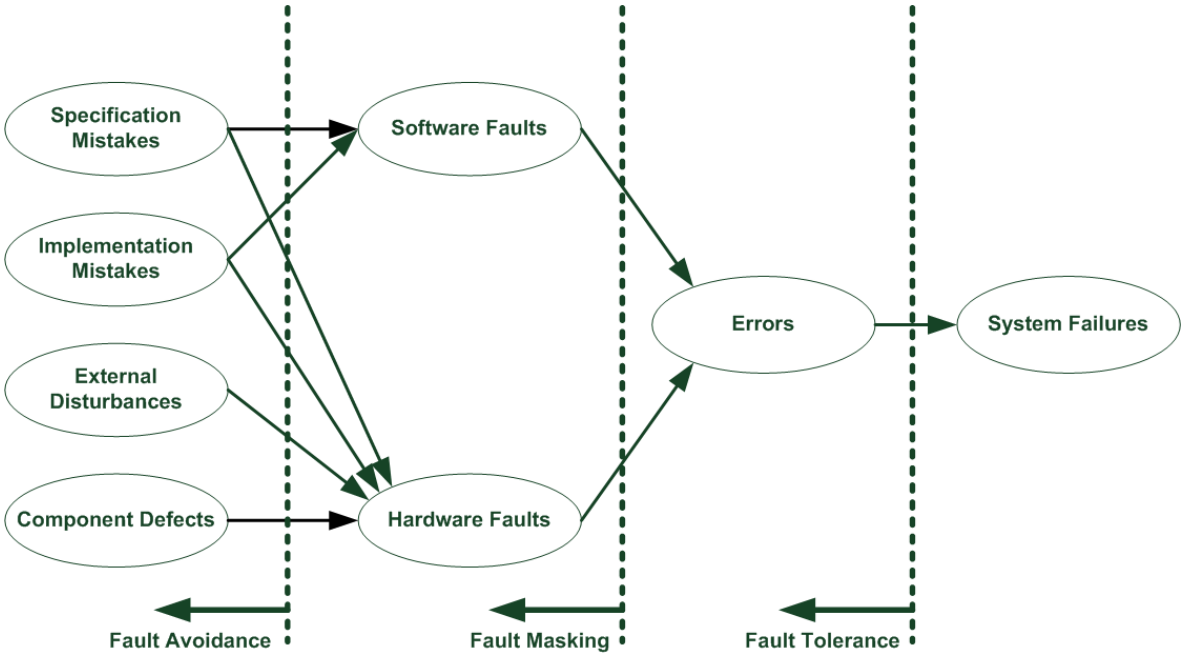

Figure 2.2: Relations among various types of Faults, Error, and failure, and Different Levels of Mitigation Techniques

Fault Avoidance:

Strictly speaking, fault avoidance is not part of fault tolerance strategy as the idea of fault avoidance is to make sure that neither software or hardware faults ever occur during the operation. Fault avoidance can be things such as design review, design rule checking, test and validation, choice of physical materials, mechanical design such as shielding, and manufacturing quality control. This includes choosing radiation hardened IC package such as Xilinx QV ceramic packages [32], which reduces chances of Single Event Upset faults (SEU) caused by radiations.

In most practical application field, engineers can follow the engineering standards established by various organizations such as IEEE, SAE, and ISO. [18][19] Those standards usually define all aspects of electronics design from design process, design rules, and test procedures. For instance the IEEE SEM-E standard [7] lays out the rules in designing hardware for space based applications, which need to withstand harsh environment in terms of temperature, mechanical stress, and radiation. Not only is it usually a mandatory requirement to comply with appropriate standard, it also helps designers to greatly enhance their fault avoidance measure as those standards are built on previous experiences of developing specific applications.

Although Fault avoidance is a very important part of design processes, the history and common sense tell that there can never be an electronics product that has absolutely zero probability of errors during its operation. Furthermore, excessive focus

on fault avoidance comes expensive; choice of expensive material, lengthy design and validation time, excessive test, and zero tolerance quality control at manufacturing plant all come at extremely high cost. Therefore, it is important that only realistic level of fault avoidance measure to be taken, and the coverage of fault avoidance is clarified so that possible errors are appropriately dealt with by run-time measures discussed in the rest of this section.

As the focus of this project is to design and develop the masking and tolerance mechanisms, and it does not involve neither PCB or application design, avoidance measure is out of the scope and will not be explored further.

Fault Masking:

As it can be seen on Figure 2.2, the purpose of fault masking is to prevent the faults from causing errors by making the faults invisible from outside the design region. This would mean dynamically correcting the generated errors.

The most common form of error masking is error correcting code such as parity check and cyclic redundancy check (CRC), which are used in communicating data inside most computer systems such as memory, register, and various kinds of data buses and data links.

When data is generated or transformed, simple error correcting cannot be applied as there is no previously calculated code to check for. In these cases redundancy is used to dynamically correct the errors. There are several forms of redundancy that can be utilized for fault tolerant systems: Hardware, Software, Information, and Time. Detailed explanation of each redundancy form is explained in [9]. This project is to take advantage of reconfigurability of Ubichip platform; making the choice the hardware.

Hardware redundancy is the oldest, and most common form of redundancy used in digital systems. Although it requires extra physical space, which inevitably increases cost, simple yet robust nature of hardware redundancy is attractive to designers. There are two basic techniques in hardware redundancy: **Passive** and **Active**. Passive technique simply masks the errors without making any modification to the system. Active techniques includes detection, location, and modification but not masking. Many fault tolerant systems can be categorized as hybrid as they use passive technique to mask, and active techniques to mitigate the situation.

Several novel techniques in hardware redundancy are explained in [1]. In this project however, simple N-Modular Redundancy (NMR) technique is implemented by utilizing Self Replication (SR) function of Ubichip.

The NMR fault masking technique is based on majority voting. Figure 2.3 shows the basic configuration of Triple Modular Redundancy (TMR). In this system three

identical modules are performing identical operations, and the voter unit chooses majority as the final output. If fault occurs in one of the modules, voter unit is able to mask the error as long as remaining two modules have correct output. TMR is the most common form of NMR configuration as more than three copies may not increase the reliability as larger hardware size means more probability of hardware faults, and the added cost in terms of space is significant. Figure 2.4 shows the circuit level implementation of a majority voter.

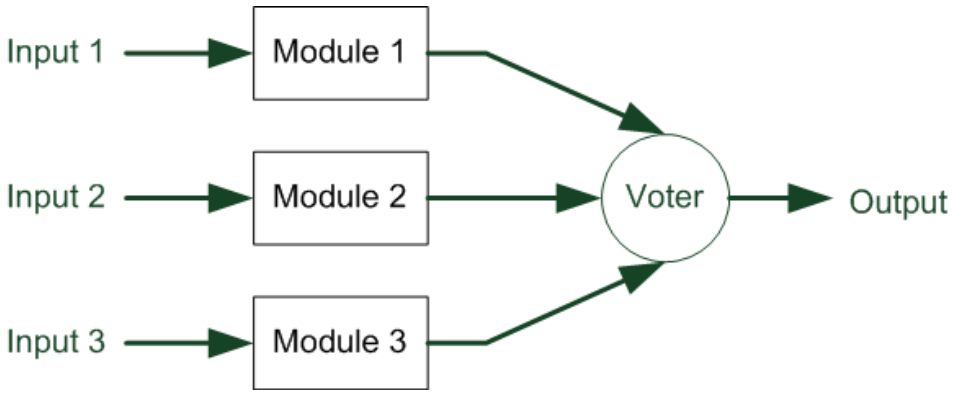

Figure 2.3: Triple Modular Redundancy (TMR) Concept

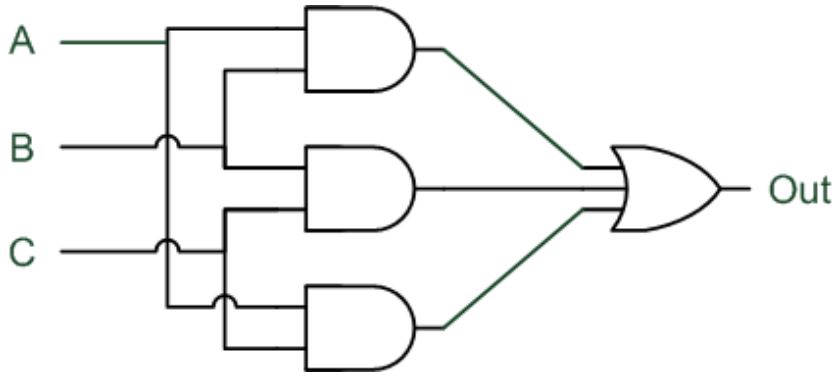

Figure 2.4: Majority Voter Circuit

Fault Detection:

Detecting fault is essential for active fault tolerance techniques. Figure 2.5 shows a simple hardware redundancy configuration that detects error. Outputs from two identical modules with identical operation are compared to detect if they agree. When the outputs from the two modules disagree, the comparator detects this and indicates the occurrence of an error. However, being a purely active configuration, there is no modification to the output.

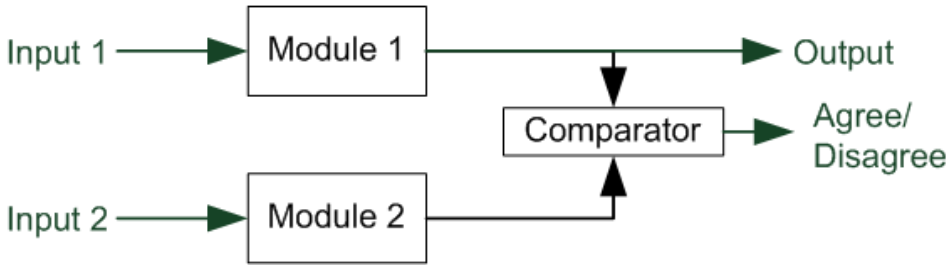

Figure 2.5: Simple Fault Detection

Fault Containment:

Containment is an action that prevents the errors from propagating outside the boundary. Masking takes care of containment dynamically. When there is no masking mechanism in use, the fault tolerance mechanism must notify relevant modules that the output is not valid. This approach may be troublesome in time critical applications such as real-time control systems.

Fault Diagnosis, repair/reconfigure, recovery:

Hybrid configuration, which mixes both passive and active fault tolerance techniques are required to not only mask but detect and mitigate the faulty condition in the system. In order for the system to be fault tolerant, it must locate the fault and recognize the extent of the fault (diagnose), replace the fault with fault-free component (repair/reconfigure), and finally bring the system to a state where continued operation is possible (recover).

Sometimes a system capable of these actions are called Build In Self Repair (BISR). Common example of BISR is a large DRAM circuit with spare memory block, which replaces a faulty block when error is detected. Figures 2.6 and 2.7 show some of the hybrid configurations.

'NMR with Spare' circuit shown in the Figure 2.6 has N number of identical modules conducting the same operation. Voter/switch unit selects more than three units for the majority voting. Output from the voter is fed back to the error detecting unit, which upon detecting an error de-selects faulty module and the 'spare' module is used instead.

In the 'Self purging' system shown in Figure 2.7, each module has its own output switch, which is comparing the module output and the system output. When the module output disagrees with the system output, switch shuts off the module output; purging the faulty module from the majority voting system.

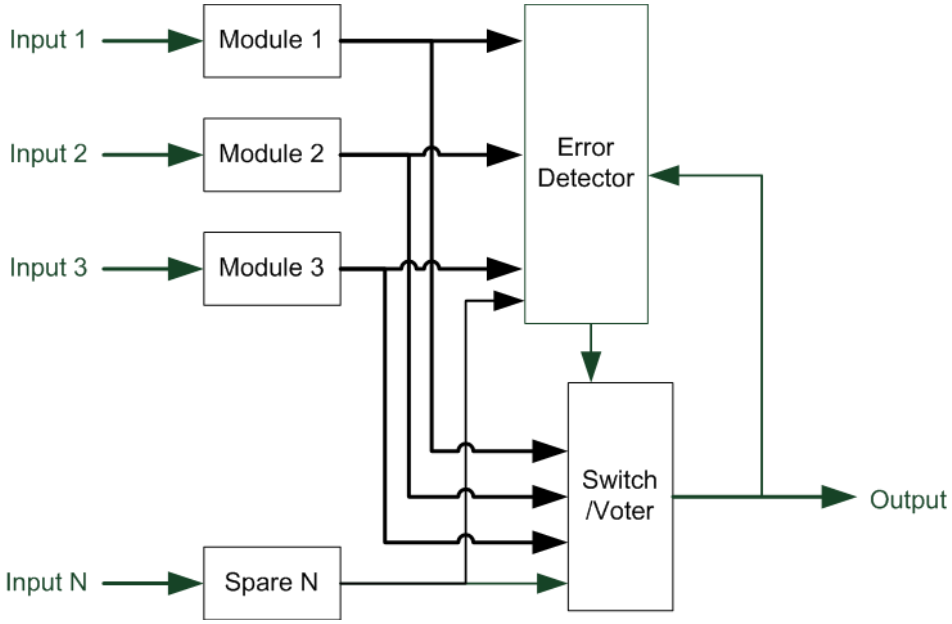

Figure 2.6: NMR with Spare

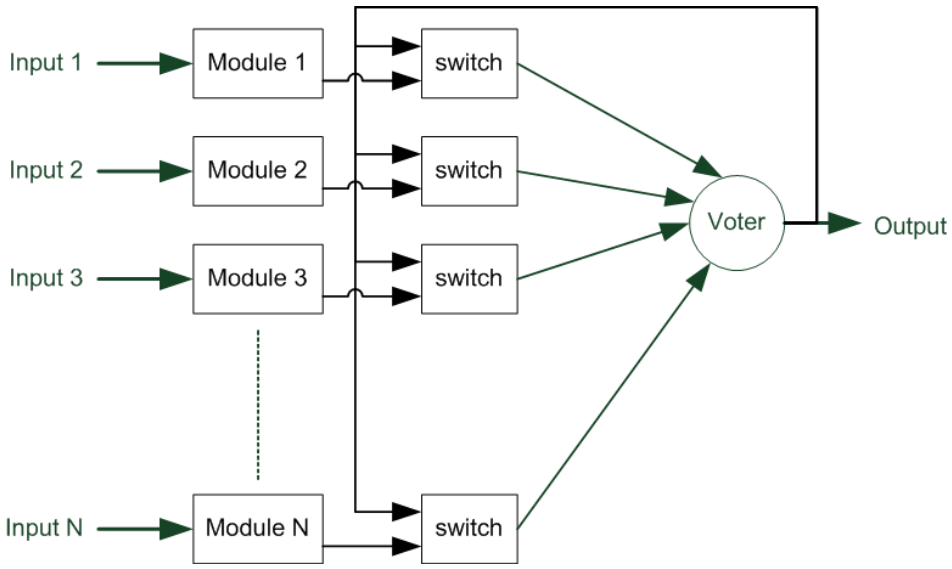

Figure 2.7: Self Purging

2.3 Power Aware Systems

In order to reduce the power consumption as much as possible, modern LSIs commonly incorporate techniques such as clock gating and power gating. Clock gating stops the supply of clock signal for unused logic circuit block so as to eliminate the dynamic power consumption caused mainly by switching current of transistors. Power gating stops the supply of power all together to completely shut down the circuit. [23]

In this project, number of functional unit copies created for Triple Modular Redundancy (TMR) is controlled dynamically to control the overall power consumption. In order to achieve such system the power consumption needs to be monitored constantly and accurately. The author looked into Built-In Current Test (BIC) techniques, where built-in analog circuit measures the current consumption of the device to find device hardware fault. Literatures [20] and [26] describes how such circuit can effectively be designed and implemented. An example of such circuit is shown in the chapter 3.

2.4 Perplexus Project: Ubichip Platform

The design of this project is implemented within a framework developed in the PERPLEXUS European project. The Ubichip is the kernel of this project; a reconfigurable VLSI system endowed with bio-inspired capabilities. As reported in [11], attributes of bio-inspired architecture is promising for implementation of fault tolerant hardware especially in the application of Build-in-Self-Repair (BISR), wherein self-healing capability of the hardware detects and mends faults in the hardware. This section explains the important functionalities of Ubichip that is used in this project. Details of the PERPLEXUS project can be found in [31], [22].

2.4.1 Ubichip

The Ubichips are mounted on a prototype system called Ubidule, which is presented [31]. As shown on Figure 2.8, Ubichip consists of three major blocks: An array of reconfigurable processing elements called Macrocell (MC), the System Manager and a controller for Content Addressable Memory (CAM). The system manager block is responsible for configuring the reconfigurable array and external communication.

Each MC is made up with four reconfigurable cells called Ubicell, which is explained later in this section. The configuration bit stream for each MC can be recovered and configured dynamically using the Self-Replication (SR) function of the Ubichip. The SR function is used extensively in this project, thus its details are briefly explained later in this section. Each MC also contains a Dynamic Routing (DR) control unit, which allows a pair of MCs to establish communication paths dynamically. The DR functionality of Ubichip is further explained in [27]. Furthermore, a Ubichip can also be configured in multiprocessor mode where a SIMD-like parallel machine can be implemented.

Initially the fabricated Ubichip was scheduled to arrive at the author's laboratory however there has been a delay and the author did not get access to the finished Ubichip.

2.4.2 The Ubicell

Figure 2.9 shows the overall organization of an Ubicell. An Ubicell consists of an input switch box, an output switch box, memory/LUT block, ALU block, and a flags

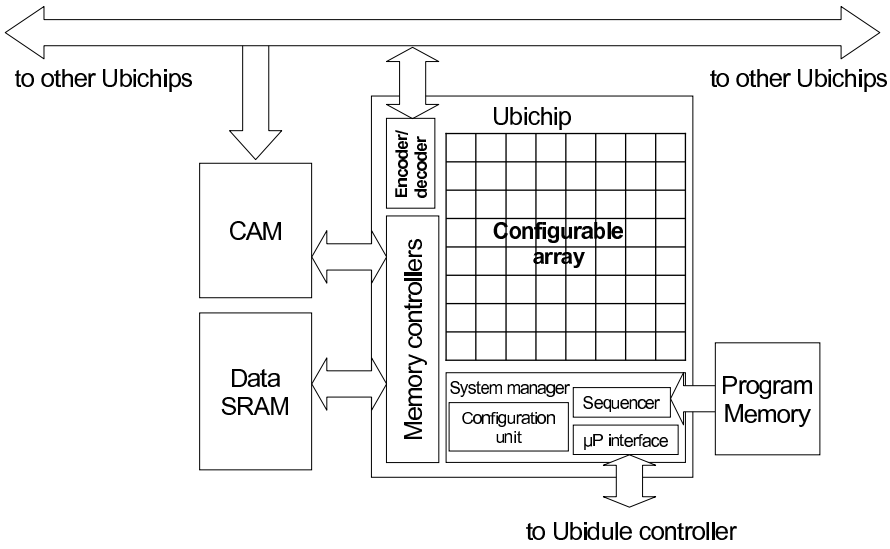

Figure 2.8: Overall Block Diagram of a Ubichip [31]

switch box. The ALU block consists of a 4-bit ALU to be used for multi-processor mode, which is unfortunately not available for LUT mode operation, which is what this project implementation is in [15]. In this project, relevant blocks are input switch box, output switch box, and the LUT/memory section.

The LUT section has four 4-bit Look-up-Tables (LUT), each of which has input and output multiplexers to realize nine different configurations listed in table 2.2. Schematics of each configuration can be found in the appendix section.

1	Four Independent 4-input functions
2	Wide Decoder/high-fanin function
3	Two-lelvel Logic
4	Counter Mode
5	Configurable Registers
6	2 x 2-bit state machine
7	3-bit state machine
8	Shift Register
9	LFSR Mode

Table 2.2: Ubicell LUT configurations

Input box selects the input for each of 4-bit input ports located at each side of Ubicell, North, East, South, and West. As shown in the Figure 2.11, input from each side can be selected from 4 of the neighbouring cells. The input switch box allocates the input for each of the four internal LUTs. The output switch box selects output from each side of the cell; there are four separate multiplexers for

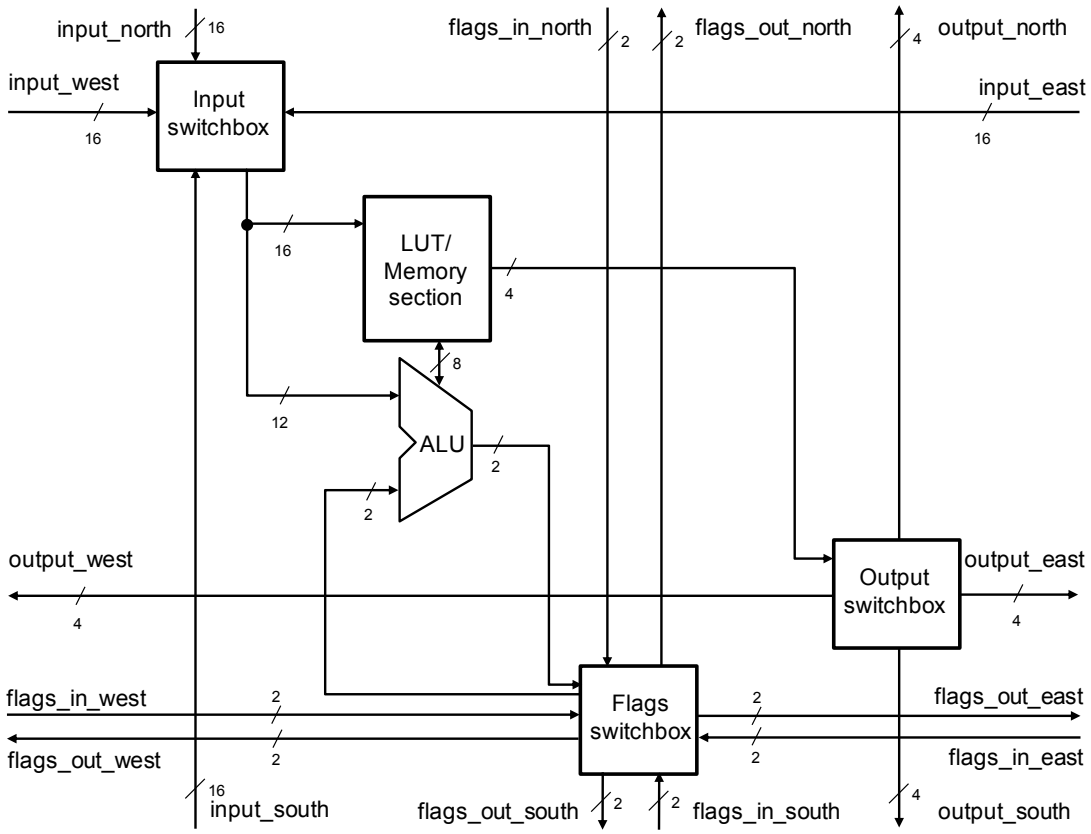

Figure 2.9: Organization of Ubicell

each side of the cell. The outputs can be selected from 16 different sources including raw signal from the input of other sides as shown on Table 2.3. By selecting input signal to other sides, signals can be routed to cells farther than the immediate neighbors.

2.4.3 Ubicell Array Organization

A **Macrocell** is mede up with four ubicells as shown in Figure 2.13. Figure 2.12 shows the block diagram of a macrocell. Macrocell is the smallest unit of cells when bio-inspired capabilities of the Ubicells are utilized, namely Dynamic Routing (DR) and Self Replication (SR). DR and SR are discussed in the next sections.

In a sense of bio-inspired architecture, an Ubicell is called a **moleule**, a Functional Unit (FU) made up of macrocells is called an **organelle**, organelle with additional circuit for self replicating (SR) capability is called a **cell**, and a set of cells in one system is called an **organism**. The size of organism on Ubicell can be up to 10x10 macrocells (20x20 Ubicells).

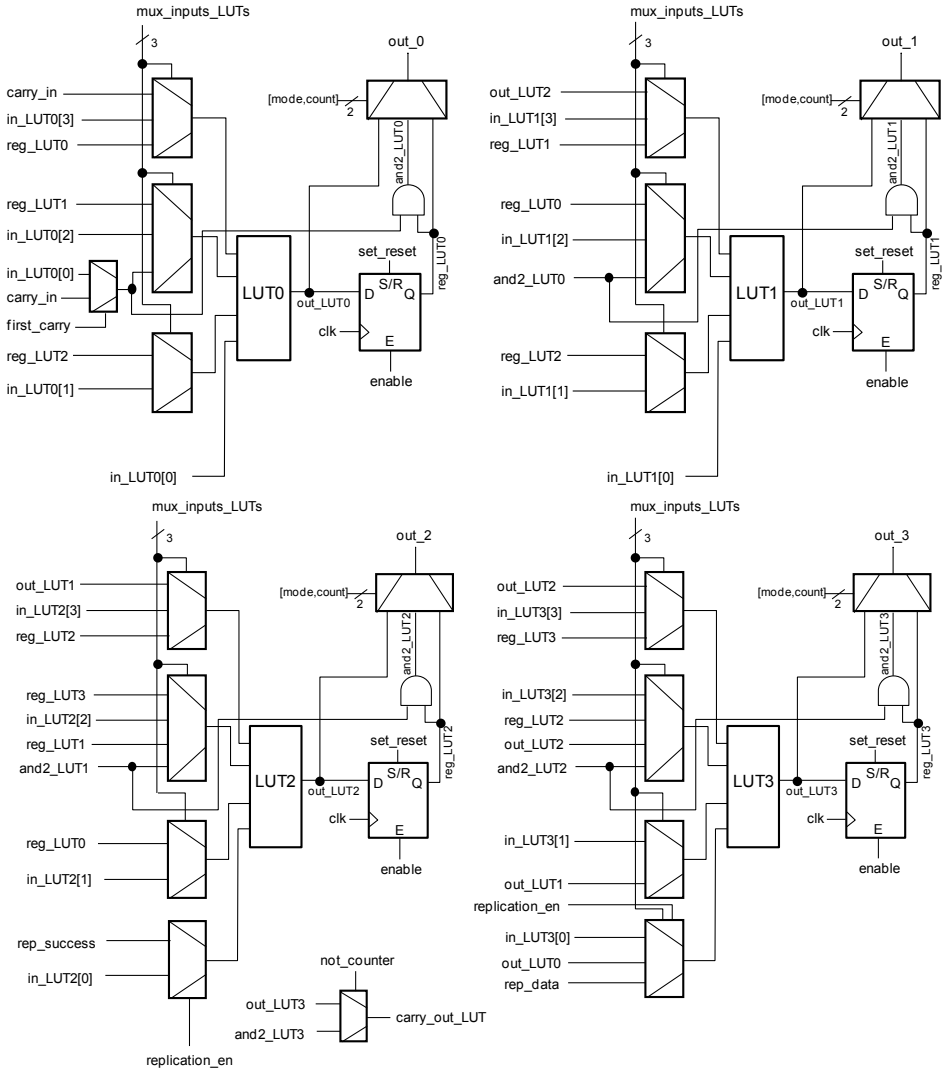

Figure 2.10: Ubichip Look-up-Table (LUT) I/O Organization

2.4.4 Self Replicating (SR) Capability

Self Replication (SR) is an important attribute for Ubichip; it allows Ubichip to implement bio-inspired circuits featuring growth, learning, and evolution. In POE model (Phylogeny, Ontogeny, Epigenesis), which is the main target of Perplexus project, SR capability allows Ontogeny, which concerns the developmental process of multicellular organisms [31]. In SR process, a cell is capable of creating exact and complete copy of itself at arbitrary location within the reconfigurable array. The SR process on Ubichip occurs by means of self-inspection; the cell inspects its own configuration bit-stream in order to make a replica of itself. While this approach eliminates the necessity of extra memory element to store the configuration data, it adds up complexity of the control circuit. As it is seen in figure 2.15, two separate SR controllers are necessary for the entire process. Furthermore, sequence of events listed below needs to be implemented

	Output Options:
1	Output from the LUT Block
2	input_south[3:0]
3	input_south[7:4]
4	input_south[11:8]
5	input_south[15:12]
6	input_west[3:0]
7	input_west[7:4]
8	input_west[11:8]
9	input_west[15:12]
10	input_north[3:0]
11	input_north[7:4]
12	input_north[11:8]
13	input_north[15:12]
14	LUT0 register
15	LUT1 register
16	LUT2 register

Table 2.3: Ubicell Output Switchbox sources (In the case of East-Out)

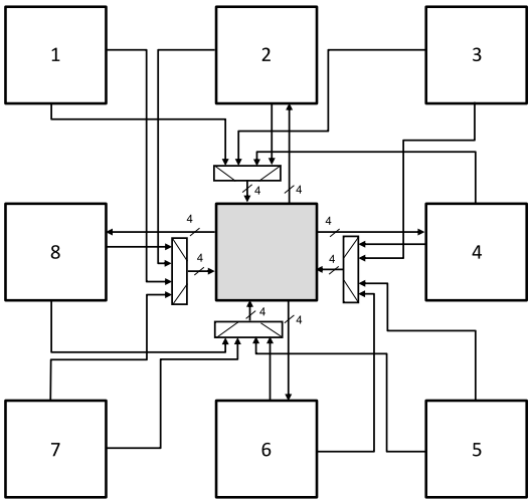

Figure 2.11: Inter-Ubicell Connectivity

using a set of finite state machine (FSM) controlling each of the SR controllers.

1. a: SR Controller (SRC) 2 recovers configuration bit-stream (conf-b) of SRC1
2. b: SRC2 remotely configures open array with conf-b of SRC 1
3. c: SRC1 recovers conf-b of SRC2
4. d: SRC1 remotely configures open array with conf-b of SRC2
5. e: SRC2 recovers conf-b of Functional Unit (FU)

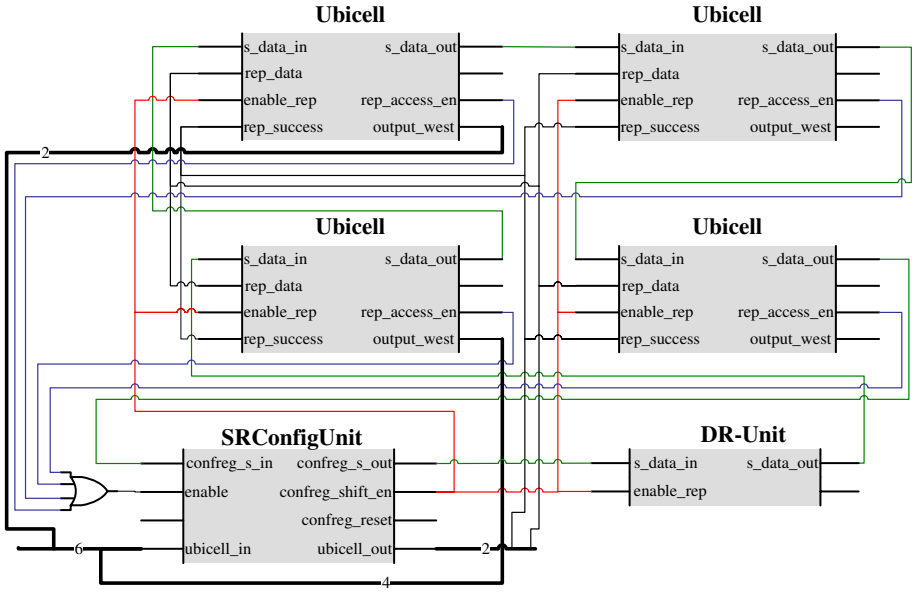

Figure 2.12: Blcok Diagram of a Macrocell

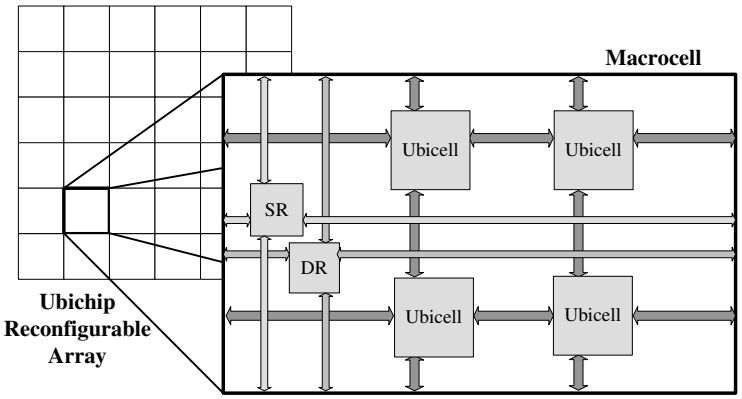

Figure 2.13: Ubicell and Macrocell

6. f: SRC2 sends the conf-b of FU to newly created crone SRC2a
7. g: SRC2a configures neighboring array with conf-b of FU

2.4.4.1 SR Mechanism

Self Replication (SR) process on Ubichip is controlled by accessing SR control units (SRCU) that sit on each macrocell (MC) (Ubicellx4). In order to do this, a MC needs to be dedicated as a SR controller by enabling SR bit on configuration. Once enabled, I/O of the LUTs in the macrocell are connected to SR control unit and SR process can be controlled through this MC. Figure 2.16 shows how a MC is dedicated for controlling SRCU, which is controlled by a FSM.

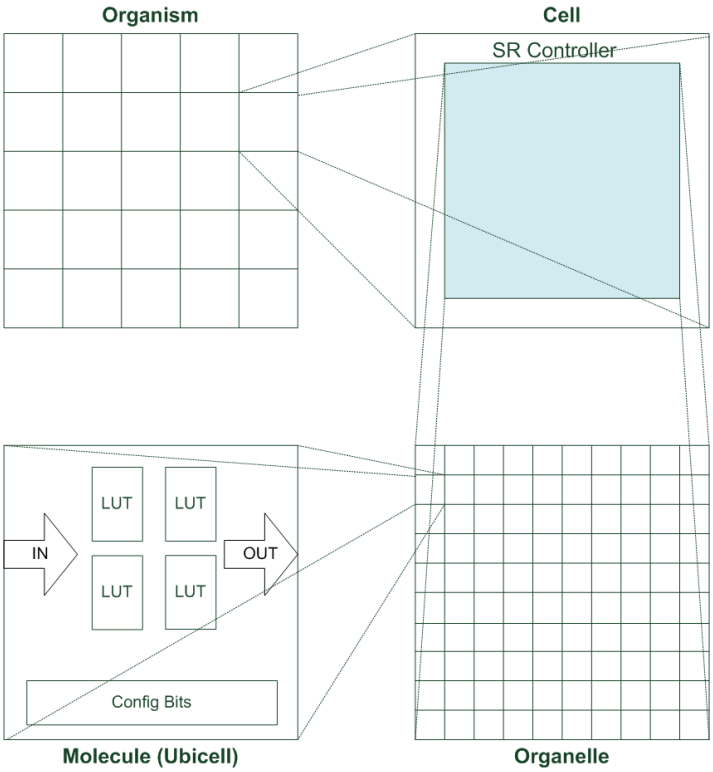

Figure 2.14: Concept of Organism

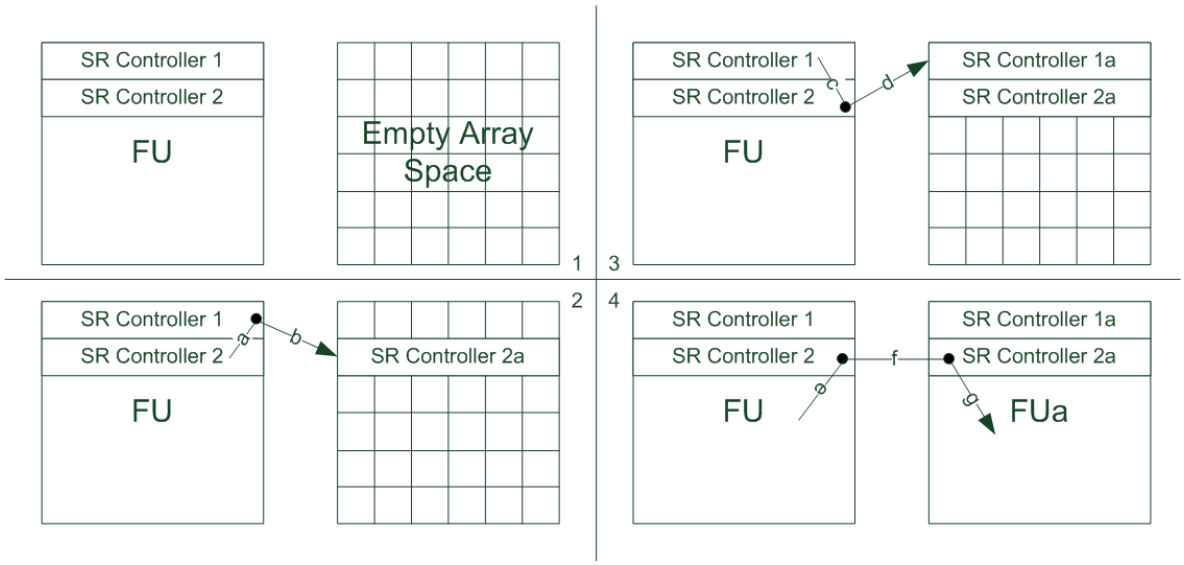

Figure 2.15: Ontogenic Self Replication Process on Ubichip

A SR control unit (SRCU) is capable of undertaking four major actions necessary for the SR process: **configuration**, **Recovery**, **Reconfigure**, and **kill**.
Configuration process involves a macrocell (MC) dedicated for the SR controller

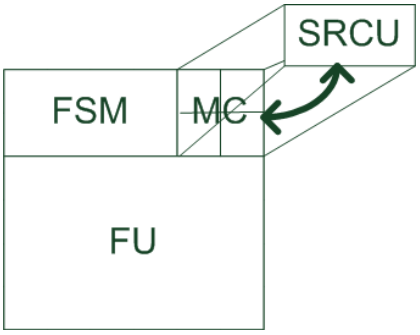

Figure 2.16: SR Control Unit and Macrocell (MC)

circuit recovering configuration bit stream (config-b) from neighboring organelle from one side (N, E, S, or W) and reconfigureing the empty cells on one of the other sides. **Recovering** and **Reconfiguring** take place when replication needs to be done on cells not at the immediate neighbor; one SR controlling MC **recovers** config-b from neighboring organelle and send it to the second MC, which is also dedicated for SR control. The second SR then configure its neighbor cells using the config-b received from the first MC. SRCU is also capable of **killing** an entire organelle, in which all the configuration of an organelle is reset.

Figure 2.17 shows the configuration bits organization of a MC. One such block consists of 525 bits. Figure 2.18 shows an organelle of 4 MCs. Configuration bits of each MC are connected serially as a chain of shift registers.

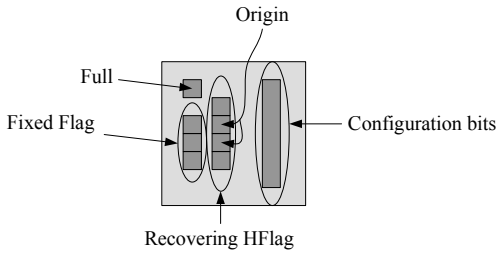

Figure 2.17: Configuration Bits Organization for a Macrocell (MC)

H-Flag:

Figure 2.19 shows H-Flags for the organelle shown in Figure 2.18. SRCU has no means of detecting shape or size of the organelle. H-flags represent the shape of organelle by simple directional arrows and a stop flag. H-Flags are used by controlling SRCU to connect all the Configuration bits of involving MCs in correct order. In Figure 2.20, one can see how H-flags are connecting all the MCs in the organelle, and consequently the configuration can occure into two directions (South and East). Finally, Figure 2.21 shows how SRCU uses H-Flag to shape the new copy of the organelle. Further descriptions on Ubichip's SR functionality can be found in the literature [29].

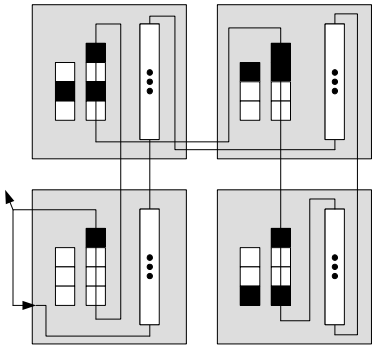

Figure 2.18: Configuration Register Chain

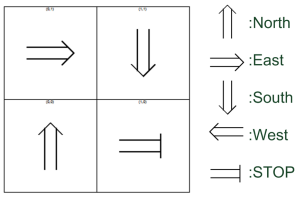

Figure 2.19: H-flag example of the Figure 2.18

2.4.5 Dynamic Routing (DR)

Ubichip has a Dynamic Routing (DR) capability, in which a pair of MCs that is not immediate neighbor to each other can be connected dynamically with 4-bit bus. As this functionality is possibly useful for this project, the mechanism was studied and

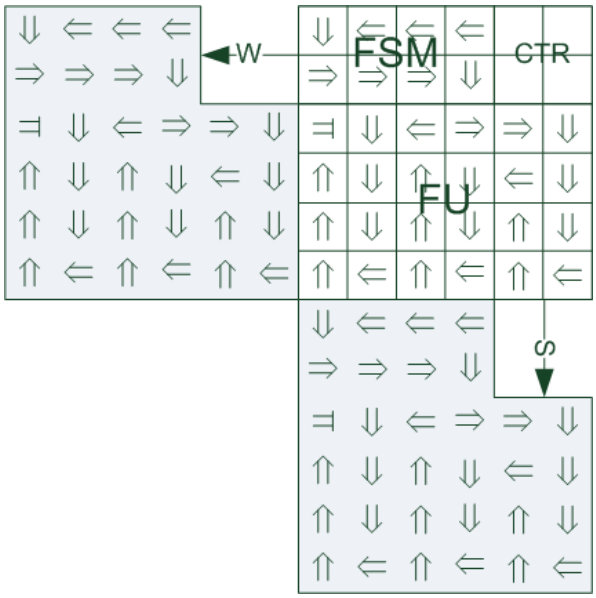

Figure 2.20: Configuration and H-Flag

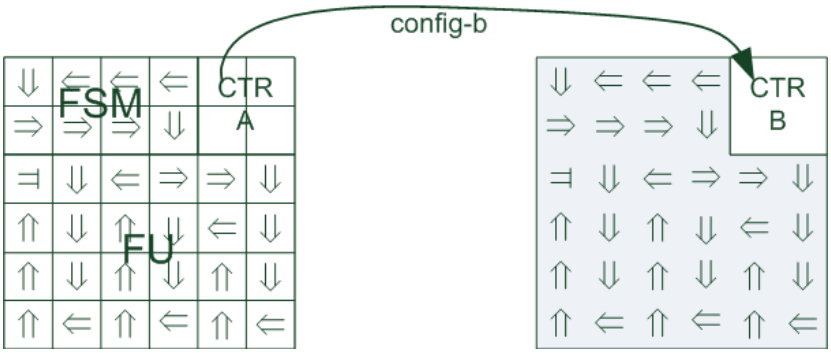

Figure 2.21: Remote Configuration and H-Flags

tested. In the end DR was not incorporated in this project but can well be used for future improvement of this system.

Similar to SR process, a DR process requires dedicated MC to control the Routing Unit (RU) as shown in Figure 2.22. Control signals, as well as the data to be sent from the source to target are connected to LUT outputs of the controlling MC. A RU has access to eight of its neighbors: North, Northeast, East, Southeast, South, Southwest, West, and Northwest.

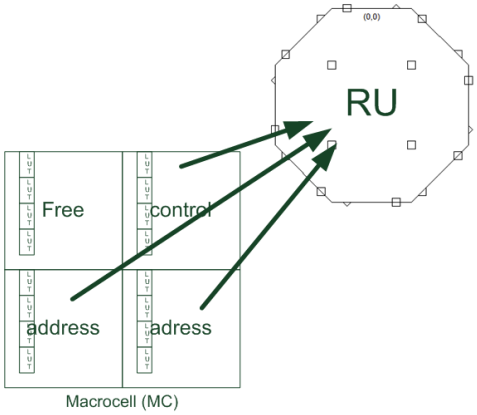

Figure 2.22: MC Controlling a RU

The RU and its controlling circuit are entirely responsible for the routing process; no external control is necessary. Furthermore, creation of path occur dynamically when a routing is initiated by controller circuit, this allows newly replicated cell to participate in the routing as well.

Only one path can exist at a time in an Ubicell array. The path is created between predefined source and target using the HIDRA algorithm, which is based on Dijkstra shortest path algorithm. The controlling circuit is responsible for destroying path once the communication is no longer needed. The path creation is done in 5 steps as shown in

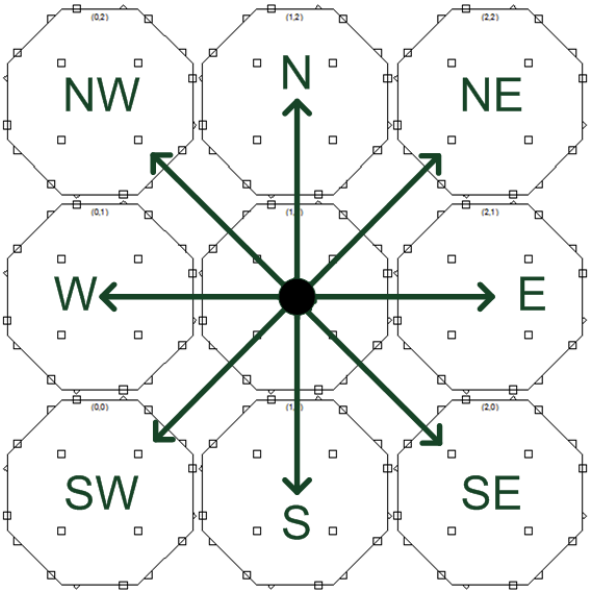

Figure 2.23: Routing Unit (RU) Connectivity

Table 2.4. The algorithm of the path creation is explained extensively in [27] and [30]. Graphic representation DR process from a simulation is shown in the Appendix section.

1	Master Allocation	Master is found based on proority rule (South West has priority)
2	Send Identifier	Broadcast the address to find the possible pair
3	Competitor Elimination	The nearest RU with the same identifying address is chosen
4	Expansion	Breadth-first search algorithm to find the path
5	Path Creation	A path is created when search is over

Table 2.4: Dynamic Routing (DR) Path Creation

2.4.6 Ubimanager

A software tool called Ubimanager was used to implement the design on Ubichip. Ubimanager was designed in the PERPLEXUS project in order to manage the Ubichips. The Ubimanager allows developers to design Ubichip implementations by means of a GUI environment; developers can configure all the three layers of Ubichip: Ubicells, Dynamic Routing Units (DR), and Self-Replication Units (SR). It is also capable of simulating the implementation using Modelsim. A detailed description of Ubimanager tool is provided in [28].

In a Ubimanager environment, the array of Ubicells is represented in a GUI window; a developer can configure each cell by double-clicking the cell to open the configuration window. Screen shot of various UbiManager GUI can be found in the Appendix Section.

3 System Architecture

After being familiarized with the architecture of the Ubichip as well as leaning about the concept of fault tolerant system design, which were explained in the previous chapter, the author has first designed a top-level system architecture; utilizing the Ubichip functions to realize a power-aware fault tolerant system. This chapter will explain the basic architectural overview of the system.

Implementation took two stages: firstly on the Ubichip framework explained in the chapter 4, and secondly using standard VHDL design environment explained in the chapter 5. The Ubichip implementation was to present a proof of concept; verifying the capability of Ubichip as well as identifying the merit and issues. A complete system was then implemented on standard VHDL environment to address the issues identified during the Ubichip implementation stage; realizing all the necessary functional blocks, some of which were not possible on the Ubichip environment.

3.1 Overall Design

Figure 3.1 shows the top level architecture of the power aware fault tolerant system. The system contains four basic components: a FMS to control the overall functionality, Functional Unit (FU) to realize the main combinational and sequential circuits, spaces where copies of FU can be made, and voter/locater block to detect and locate errors.

3.1.1 Basic Operation

The system contains a functional unit (FU), where the main functionality of the system is implemented. There are number of spaces, in which the circuit of the original FU can be copied to realize error detecting function by means of Triple Modular Redundancy with spare(TMR with spare). An external circuit measures the dynamic power consumption of the system and feed the value through the Power-mode input, the system takes the power consumption into account when creating copies of FU; when the power consumption is below the predefined threshold required number of FU copies are made to achieve the highest reliability while higher than threshold current consumption makes the system to limit the number of FU copies to make sure that the overall power consumption of the system does not exceed the predefined limit. The system has Built-in-Self-Repair capability; any FU block with erroneous output can be reconfigured, or a new FU copy is created to replace the dysfunctional block. In a case where system detects an error that cannot be corrected, the system notifies the external controller through 'fatal-error flag'.

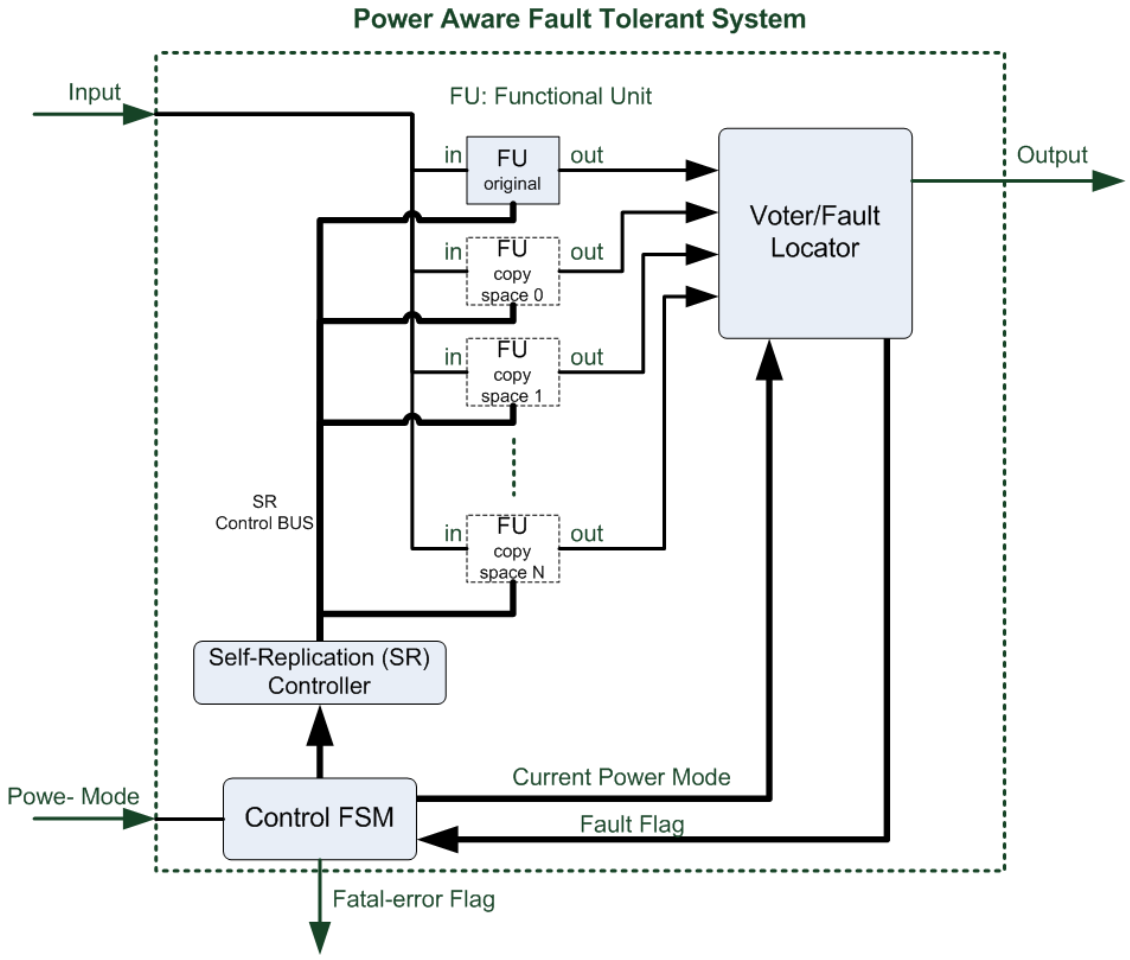

Figure 3.1: Top Level Architectural Block Diagram

3.2 System Requirements and Algorithm Design

The major functional requirements and algorithms of the system were to be considered at this stage of the design:

- Fault Tolerance
- Self Repairing
- Power Awareness

3.2.1 Fault Tolerance

TMR with spares configuration was selected as the main architecture because the Ubichip platform has a dynamic self replication capability, which makes TMR implementation simple while unused logic space can be used as spare.

Figure3.2 is a block diagram of the fault tolerance part of the system. Fault is masked by TMR majority voter circuit. Output from each Functional Unit (FU) is compared with the voter output by simple XOR circuit for fault locating. There are also three voters for redundancy and disagreement among the output of voters results in the fatal system error.

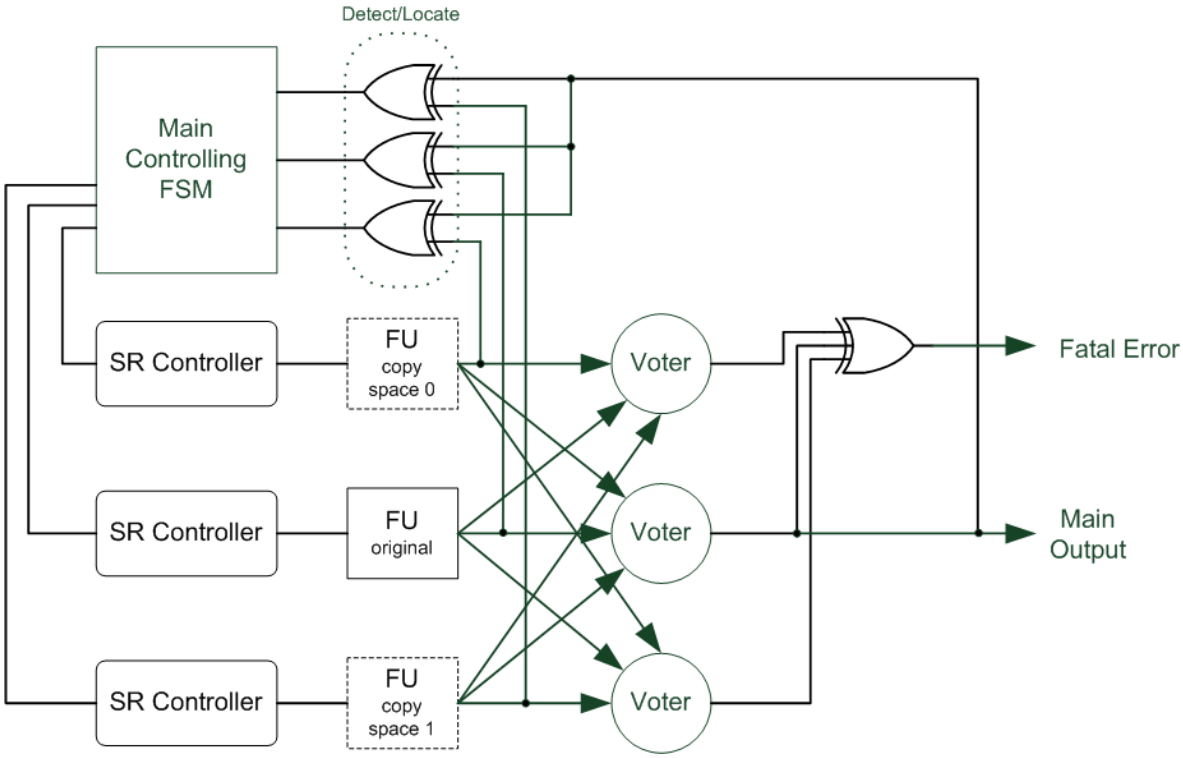

Figure 3.2: Fault Tolerant Circuit Block Diagram

3.2.2 Self Replicating, Dynamic Configuration, and Self Repairing

Figure 3.3 shows a block diagram of the Self Replication (SR), which is primarily used to make copies of the Functional Unit (FU) in order to achieve TMR fault tolerance. SR controllers are built in circuit of Ubichip, which can read (recover) and write (configure) the configuration bits of reconfigurable fabric (Ubicell).

Each of replicating block require one SR controller; this system require one for the original FU for recovering, one each for the rest of TMR copies. When system replicates the FU, SR controller for for the original FU recovers the configuration bits, which are sent directly to the other SR controllers for them to configure the empty space to make FU copies. The Self Replication mechanism of the Ubichip is serial; making a temporary termination of system operation while SR process is in progress. SR controllers themselves do not have control logic circuit and extra logic circuit is necessary to control the SR process. A Finite State Machine (FSM) is designed to

control all the SR controllers in this system.

On Ubichip, a SR controller is capable of handling variable number of reconfigurable blocks (Macrocell); size of a FU can be anything from single cell to dozens of cells. However, there is no mechanism on Ubichip for SR to dynamically tell the size of the block it must recover and replicate. Therefore a SR Timer must be implemented for the control FSM to stop the SR process after appropriate number of clock cycles. The flexibility in size of replicating block however allows the system to utilize the same space to be used for different circuit blocks.

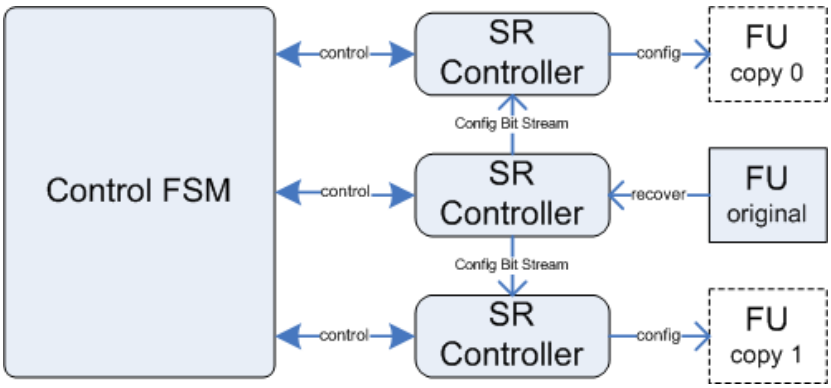

Figure 3.3: Self Replication (SR) Block Diagram

State Saving:

When copying a logic circuit dynamically, state saving is a very important feature. The circuit as well as the state of the circuit need to be copied so that the newly configured block can continue the process that was in progress before the replication.

For the system of this project to function, state saving is an essential feature because the number of TMR copies are changed dynamically to meet the power consumption specification. On Ubichip, state saving is taken care of in a normal SR process. All the circuit configuration as well as the outputs of all the LUTs are part of the configuration bit-stream. Therefore when SR process is complete the state of the circuit is also replicated.

3.2.3 Power Awareness

One of the main goal of this project is for the system to be power aware; the system must constantly be monitoring its power consumption and assure that it does not exceed the predefined threshold.

Table 3.1 and Figure 3.4 show the modes of power aware operation. Three basic modes are introduced to make this Triple Modular Redundancy (TMR) system power

aware.

When power consumption is below Threshold A (Mode 1), the system operates in normal TMR configuration, which consists of a original Functional Unit (FU) and its two copies. In Mode 2, where a power consumption is higher than the threshold A but lower than B, one copy of the TMR system is eliminated; a fault tolerance function is sacrificed but error detection is still possible. When the power consumption is above the threshold B (Mode 3), system eliminates both of the FU copies so that power consumption is kept below the limit by sacrificing fault-tolerant and error-detection capability.

Mode	Power Consumption:	Operation:
1	< Threshold A	Normal Fault Tolerant Operation (TMR)
2	Threshold A < $\bigcup$ < Threshold B	Reduced Fault Tolerance (1 FU copy instead of 2)
3	Threshold B <	No Fault Tolerance (Original FU only)

Table 3.1: Power Aware Operation Modes

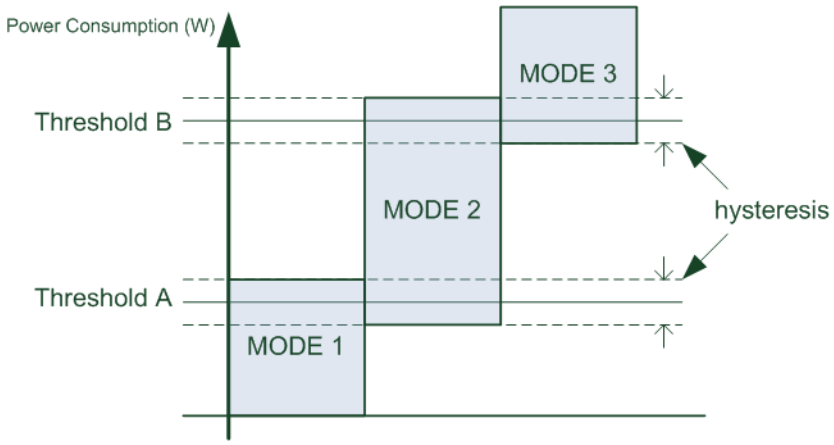

Figure 3.4: Mode of Operation and Power Consumption

3.2.3.1 Measuring the Power Consumption

Unfortunately the current version of Ubichip does not have a built-in capability of measuring the power consumption directly. In order to measure the power consumption on Ubichip framework the author used the 'transition counter', a circuit that counts the output transition of the Functional Unit to estimate the current consumption. For the proof of concept design as this project is, using the count of transitions at the output is sufficient so the system can be tested. However, for more accurate measurement of the power consumption, the author recommends implementing the analog current measurement circuit used for Built-In Current Testing (BIC Testing).

Current Measurement Circuit:

As mentioned earlier, Ucichip does not have capability of accurately measuring the power consumption. In order to make a system to be power aware, the author recommends implementing a on-chip circuit to measure the power consumption. A simple current measurement circuit designed for Built-In Current Testing (BIC Testing) can be used to measure the current consumption in relatively simple CMOS based design with small overhead. The details of BIC Testing circuit can be found in [20] [26].

Figure3.5 shows a block diagram of proposed current measurement design based on BIC circuit. An I-V converter connected serially to the system converts the current into voltage as shown in Figure 3.6. The voltage then can be digitized by simple A-D Converter (ADC) as shown in the Figure 3.7. The digitized value of current consumption should be compared to the thresholds with hysteresis implemented by the last block, where the power mode is calculated depending on the average current consumption within defined time period T.

Circuits shown in Figures 3.6 and 3.7 should work as they are shown. However, analog design techniques such as capacitive reference voltage, and clocked comparator should be used to reduce the power overhead of this circuit.

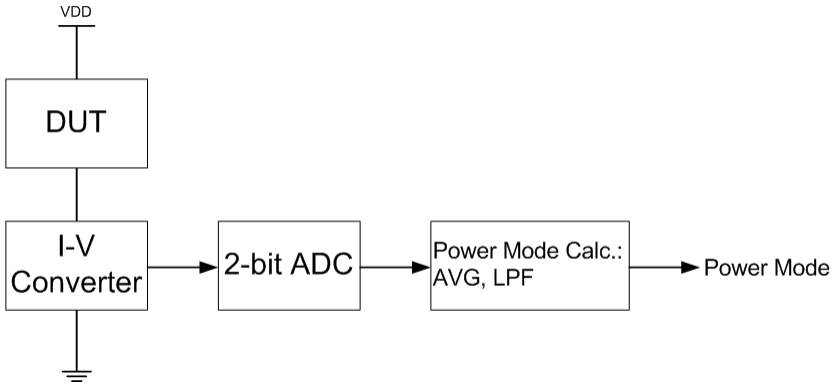

Figure 3.5: Current Sensor Implementation

3.2.4 System Controller FSM

Figure 3.8 shows design of the controller finite state machine (FSM). The system start with a single Functional Unit (FU). Depending on the power mode provided by the external power measurement unit, the FSM controls number of TMR copies to keep the power consumption within the predefined threshold. When the number of FUs are two or three, fault tolerant functionalities are valid and disagreement of the output among FUs will trigger the error mitigation actions by moving to 'error_detect' state.

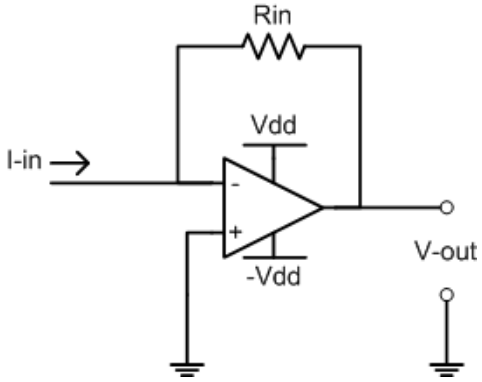

Figure 3.6: I-V Converter Circuit

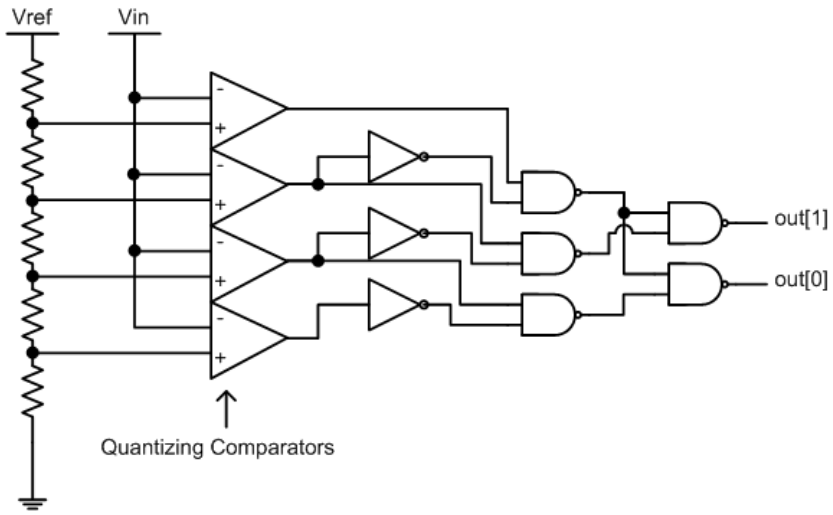

Figure 3.7: 2-Bit ADC Circuit

State:	**Function:**
reset_state	System Reset
one_FU	Initial operation state. Remain here when power mode is '3'.
two_FU	Two FU operation mode, Fault detection available.
three_FU	TMR mode, full fault tolerance available.
replicate	Reacting to the change of power mode; add or eliminate FU copy(s).
error_detect	Fault detected in mode '1' or '2'.

Table 3.2: FSM States and functions

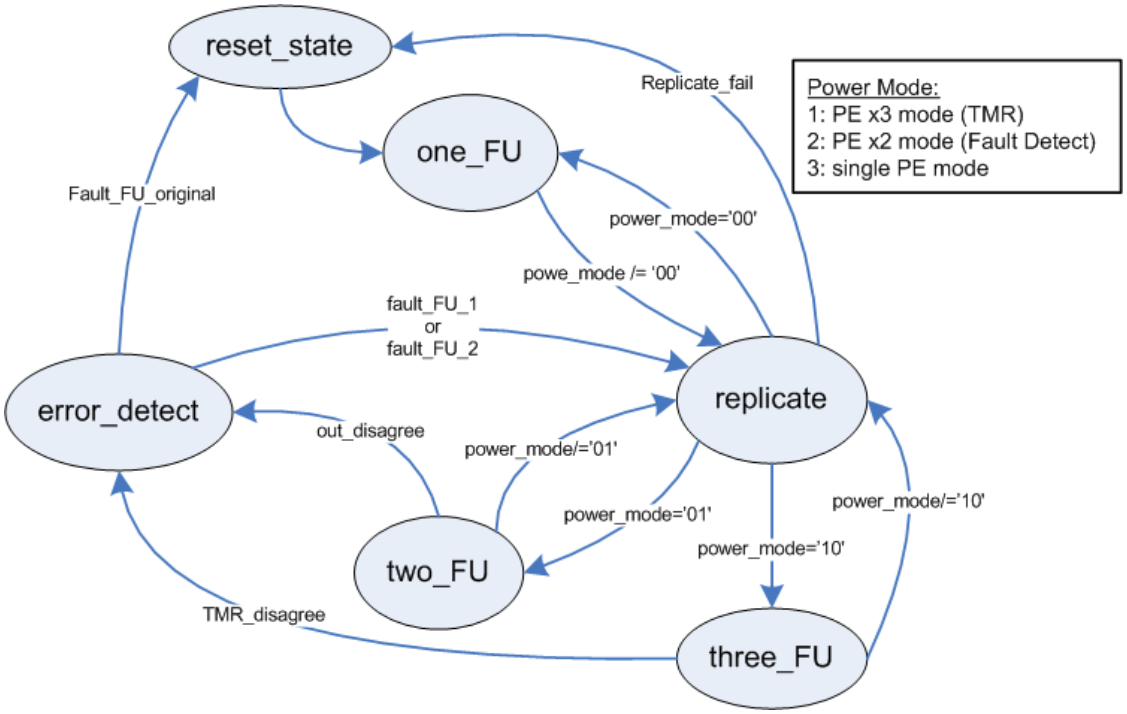

Figure 3.8: State Diagram

4 Implementation on Ubichip

The implementation on Ubichip tool place on Ubimanager development environment. As explained in the previous chapter, Ubimanager allows designers to configure the Ubichip cells in GUI environment. However, the Ubimanager does not provide compiling from high-level language or RTL design-like debug environment. Limitation on the development environment affected the design process as well as the design itself. This chapter explains how the design was implemented on Ubichip.

4.1 Overall Design

Figure 4.1 shows a block diagram of the dynamic fault tolerant system implemented on a Ubichip. There are three SR controllers; one is responsible for reading the configuration bit-stream from the original FU, being the other two responsible for replicating the copies. The control signals for the SR controllers are created in the 'Control FSM' block. The level of system power consumption is sent to the control FSM from the 'Transition Counter' block as 'counter value'. According to the power consumption level the FSM changes the operation mode and forces the SR controller to have an appropriate number of FU copies. Every time new copies of the FU are made, the Control FSM block relies on the signal from the 'SR Timer' block to stop the SR process upon completion. The outputs from FUs are compared at the 'Output Comparator' block. Figure 4.2 shows the overall system implemented on Ubichip.

4.2 Design on Ubichip

Designing a system on Ubichip consists of defining five major items in each of the Ubicells:

- Contents of LUTs
- Input select for LUT input MUXs
- Selection of the Output signals
- Selection of the Input signals
- Selection and setup of LUT configuration

As explained in Chapter 2, inputs and outputs of the four LUTs inside each Ubicell can be selected from different sources depending on the LUT configuration modes.

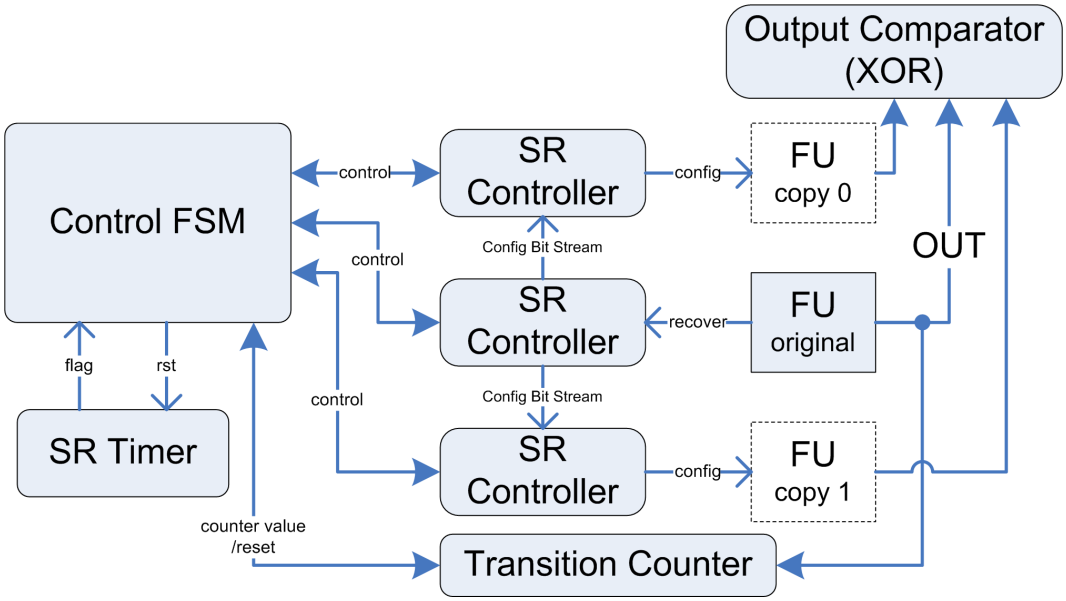

Figure 4.1: Power Aware Fault Tolerant System: Overall Block Diagram

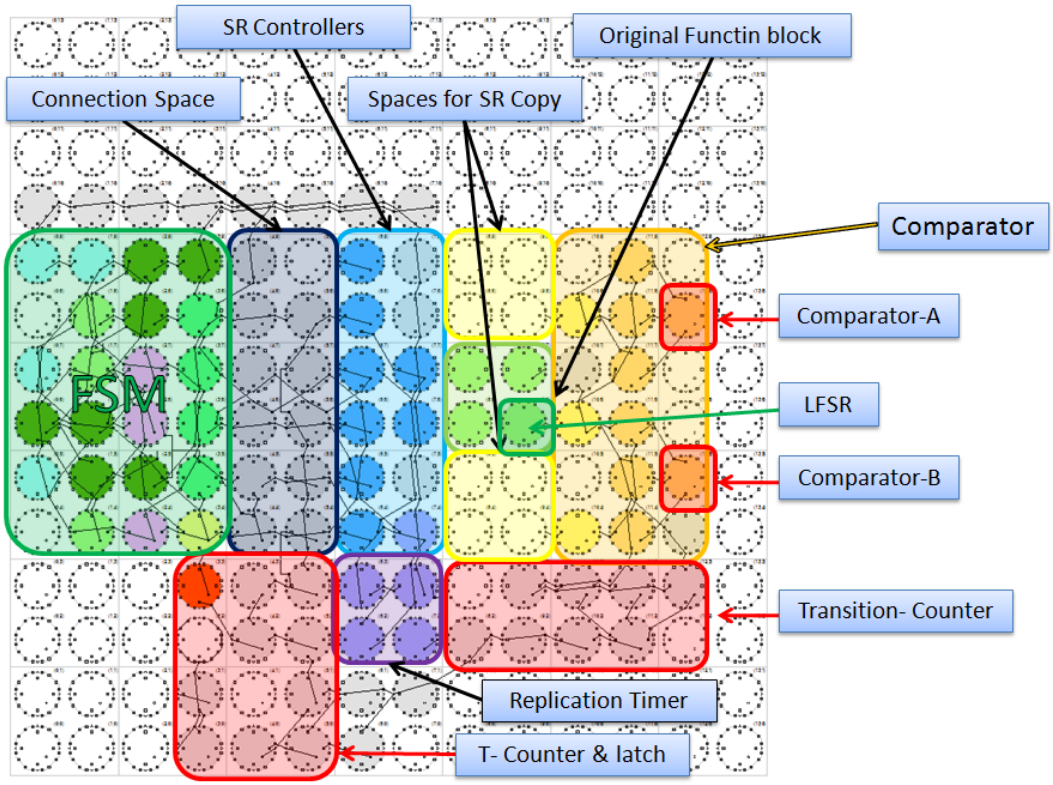

Figure 4.2: Overall System View of Ubicell Array

There are eight such configuration modes, all of which are shown in the appendix section on page 55. Definition of those Ubicell configurations are conducted using a cell configuration window of the Ubimanager as shown in figure 4.3, and the design information for all the Ubicells in a design is saved as Ubicell File (.ubc). Detailed explanation of LUT configuration modes can be found in [14].

Routing and Floorplaning:

All the routing and floor planning are conducted manually. Routing is done by defining the input and output of each cell. The routing has a significant limitation as only one 4-bit wide signal can be mapped for I/O on each edge of a cell. Without any automatic tools, planning of the location of each functional cells, connections among cells, and overall floor-planning are very crucial and time consuming parts of the design implementation on Ubichips.

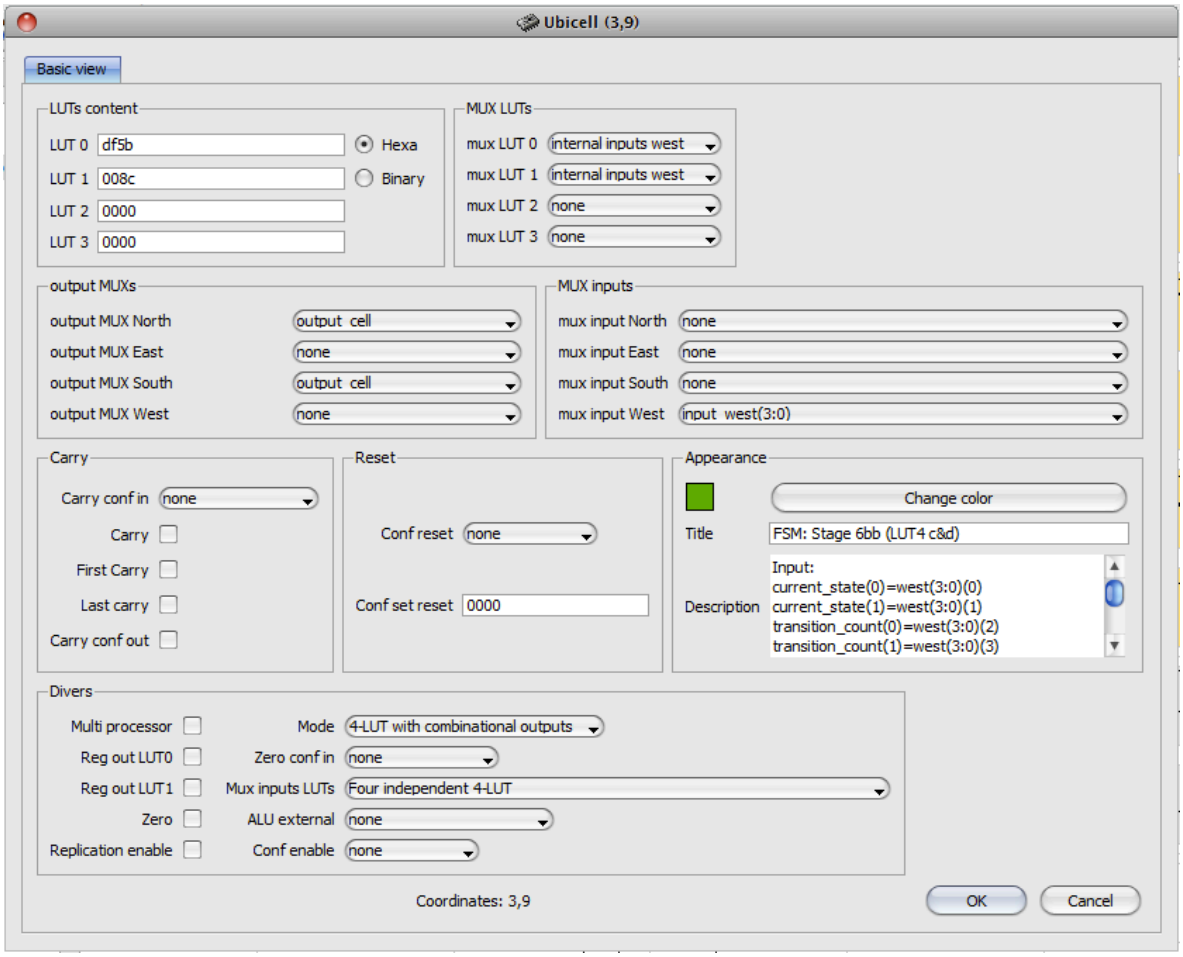

Figure 4.3: Ubicell Configuration Window on Ubimanager

4.2.1 FSM Design

LUT based architecture of Ubicell allows simple implementation of logic cicruits; having 16-bit (4x4) input and 4-bit (1x4) output, logic circuit can simply be mapped to the LUT as it is seen on the truth Table. Selecting a right LUT configuration mode, it is also possible to implement multi-stage logic and simple state machines. However, when the complexity of the logic circuit to be implemented becomes larger than a few signals, it becomes highly difficult to implement a system using only what is available on Ubimanager tool. The system the author has designed in the previous chapter is neither large nor complicated if conventional high-level environment such as VHDL was to be used. However, having 6 states, more than 20 FSM signals, and multiple control signals makes this implementation rather complicated.

The author initially intended to implement the FSM manually using classic truth-table and Karnaugh Map approach however quickly realized it is extremely time consuming and error prone task. Next the author conducted some research on techniques regarding LUT implementation of FSM systems. There are numerous papers such as [12] describing algorithm for converting FSM into LUT map. However, information the author found were all for algorithms for Electronic Design Automation (EDA) systems such as compiler and not for manual implementation.

In the end, some functions of conventional EDA tools were used to aid this implementation. Two software packages were used:

- Mentor Graphics HDL Designer Series, HDL Designer
- Mentor Graphics Precision RTL Synthesis 2008

From the HDL Designer software, a function to generate a synthesizable code directly from a FSM State Diagram was used. Precision RTL tool imports the generated code from HDL Designer and map the code to selections of FPGAs. In the process of circuit mapping the Precision RTL generates a schematics with LUTs. The author used this schematics to implement the FSM system on the Ubichip. Limitation such as Ubicell array size and signal routing constraints meant that the author needed to simplify the design. Figure 4.4 shows the state diagram implemented on Ubichip. Full-size output from the EDA tools can be found on the Appendix section at page 59.

A schematic of the FSM generated was analyzed and divided into 6 stages as shown in the figure 4.5.

4.2.2 SR Controller

While it is possible for one SR controller to remove and make a copy of an organism (set of MCs), 'remote configuration' explained in [29] is necessary to control two different copies separately. In this case, a total of 3 SR units are required: one for recovering

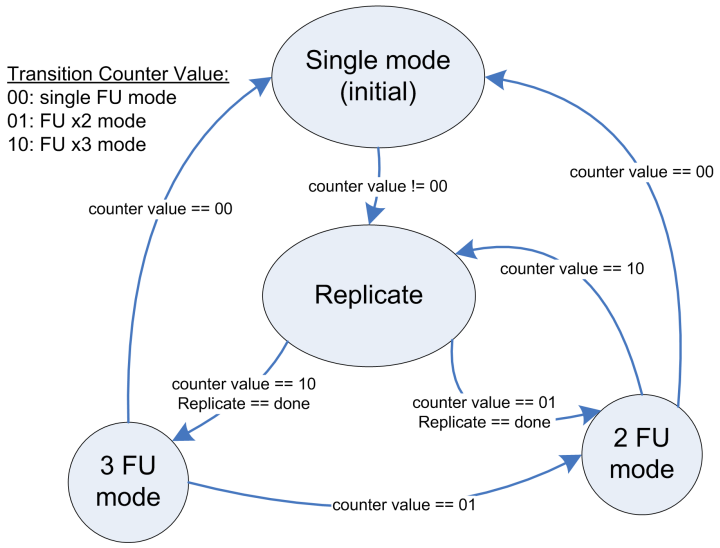

Figure 4.4: FSM State Diagram of the System Implemented on Ubichip

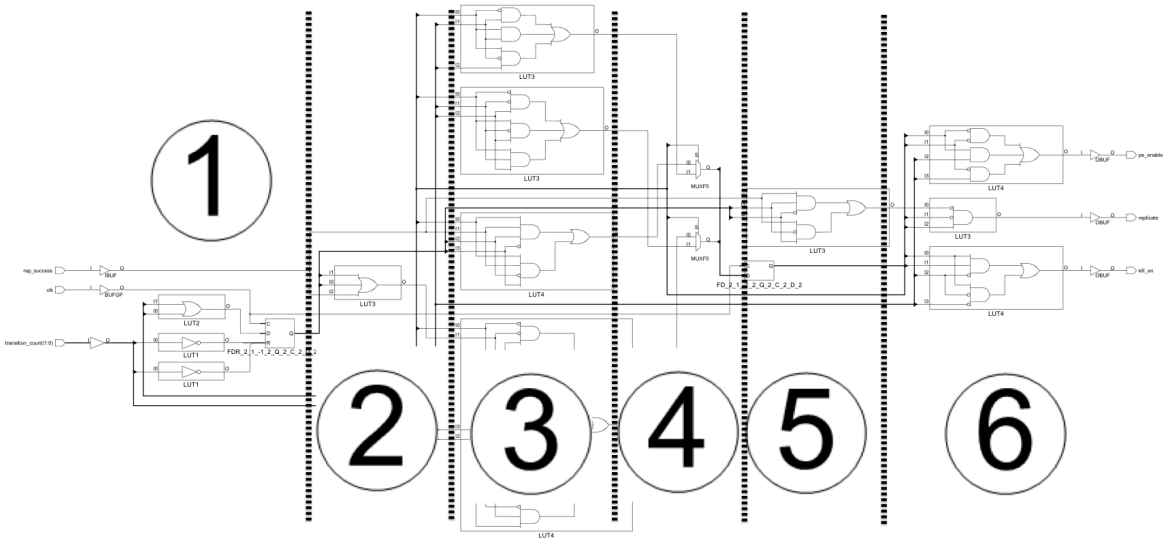

Figure 4.5: FSM Schematics based on LUT with 6 Logic Stages

the configuration bit stream of the original organism, one each for configuration of the two copies.

A SR controller takes one whole macrocell (4 Ubicells). Enabling the 'Replication enable' signal, the macrocell becomes an interface to the SR unit of neighboring macrocells. Control signals for the SR processes as well as the bit-stream data for replication are read-out and fed through the SR controller.

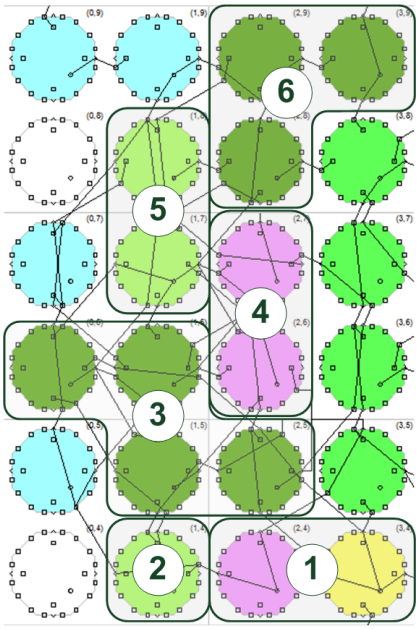

Figure 4.6: Cells with FSM Implementation

4.2.3 SR Timer

A 4-bit flag called 'H-flag' contained in each MC defines the shape of an organism. The SR unit does not have the number of MCs included in a single organism; it is not possible for the SR unit alone to determine the number of cycles required to complete a SR process. A counter is necessary to stop the SR process at an appropriate time.

A combination of 4-bit counters, comparator, and control circuit was implemented in five ubicells as shown in Figure 4.7. An Ubicell can be configured as a 4-bit counter when configuration mode shown in Figure 4.8 is selected. The comparator circuit compare the counter output from each cell, when the output reaches the predefined value, a flag is sent to the FSM to stop the SR process. The reset circuit monitors the FSM state and keep the timer from running while SR process is not in operation.

4.2.4 Transition Counter:

Because Ubichip does not have a current measuring capability, a transition counter was implemented in the system to estimate the power consumption by monitoring the logical transition of the output.

Figure 4.9 shows the diagram of 4-bit transition counter used in this project. As one can see in this Figure, a parallel counter is required in this circuit. The design of parallel counter can be complicated and require large space. In this project, the author limited the output width to 4 bits, and implemented a 4-bit parallel counter using four 4-bit LUTs as shown on Table 4.1.

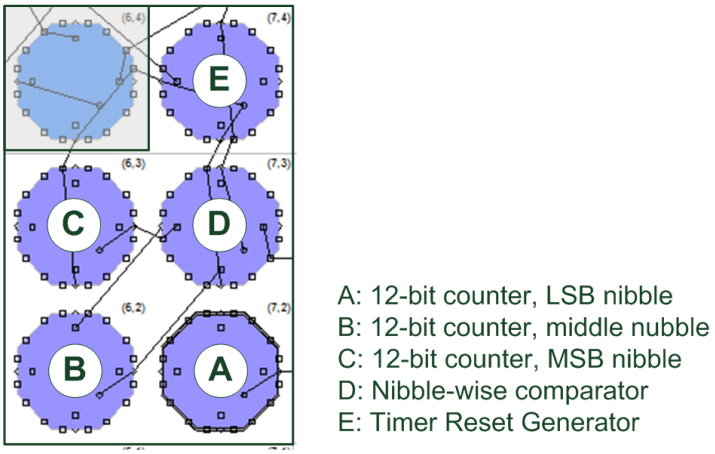

Figure 4.7: SR Timer Circuit

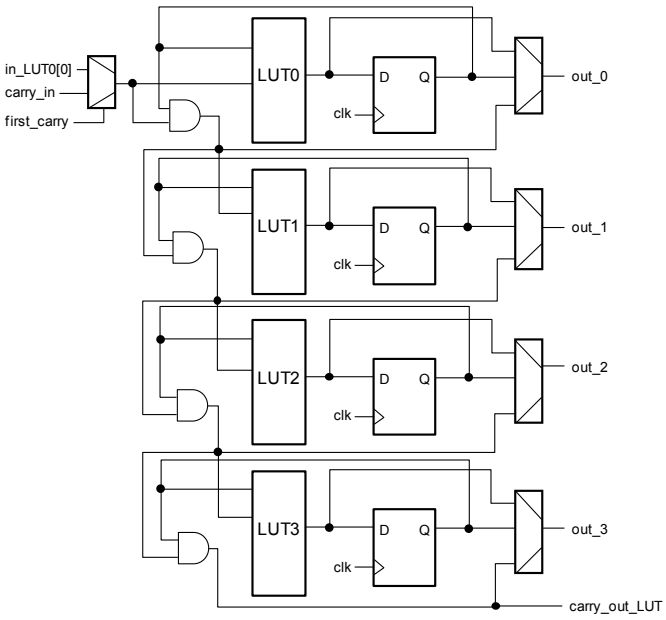

Figure 4.8: Ubicell in Counter configuration

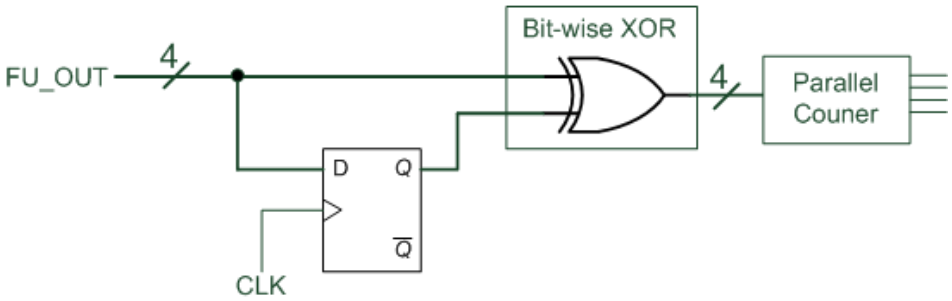

Figure 4.9: 4-bit Transition Counter

LUT Input:	Out(bin)	Out(dec)
0000	0000	0
0001	0001	1
0010	0001	1
0011	0010	2
0100	0001	1
0101	0010	2
0110	0010	2
0111	0011	3
1000	0001	1
1001	0010	2
1010	0010	2
1011	0011	3
1100	0010	2
1101	0011	3
1110	0011	3
1111	0100	4

Table 4.1: Implementation of 4-bit Parallel Counter Using 4 x 4bit-LUT

4.2.5 Functional Unit

As the goal of this experiment is to show a proof of concept working system, the implementation of the Functional Unit (FU) was kept simple; a combination of memory, a counter, and a Linear Feedback Shift Register (LSFR) pseudo-random number generator were configured in a MC as shown in Figure 4.10. A built-in configuration, which configures the LUT and the internal memory element as a 64-bit LFSR was used to implement the LFSR cell.

4.2.6 Comparator

In this implementation, output comparator simply indicates the bit-wise XOR of the outputs from each FU as a fault detecting measure. Ubichip blocks outputs from cells

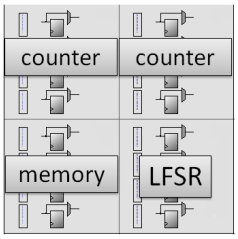

Figure 4.10: Functional Unit (FU)

during SR process so that no error is detected during normal SR process.

4.2.7 State Saving, Enable Signal

Output from each LUT on Ubicells are part of the configuration bit-stream on Ubichip. Therefore it is not necessary to implement extra circuit for copying state before the replication. However, as state saving depends on the data copied in the SR process, even the organism that is not part of the SR process must stop their operation while replication is in progress. This situation occurs in this project when the number of FU copy is increased from 1 to 2; the already existing copy of FU must stop its operation while the configuration is recovered from the original FU to generate the second copy.

In order to keep the state of all the FUs in sync, an global enable signal was introduced. The signal is generated from the FSM. Specification in the Ubichip only allows the enable signal to be connected to specific bit at West or the North input ports, this makes the implementation of enable signal complicated, and may pause problems when the size of FU is larger.

4.2.8 Debugging on Ubichip

Debugging stage was the most time consuming and difficult part of the implementation solely due to the lack of EDA tools. Errors exist in any part of the implementation though there was no tools to detect any of the errors. Design would simulate with no problem but simply would not operate correctly. Tracing the signal is not an easy task either; signals are passed through numerous cells' input and outputs but there was no means of organizing different signals as netlists. Essentially, in order to debug a system on Ubichip the designer must go go over the every configuration in each cells.

This design on prototyping environment makes a design process different from ones using conventional tools. First stages of the design flow before the actual implementation should be conducted very carefully as design change after the implementation takes almost the same time as a implementation from the scratch.

4.3 Functional Test

The implementation was tested using the Modelsim tool integrated in the Ubimanager environment. Figure 4.11 and 4.12 show screen-shots of the system under simulation. One can see how a different counter value results in a different number of copies. Each system block was confirmed to be working according to the design intention. After the system was verified by simulation it was physically implemented in the Ubichip available in the Ubidule board.

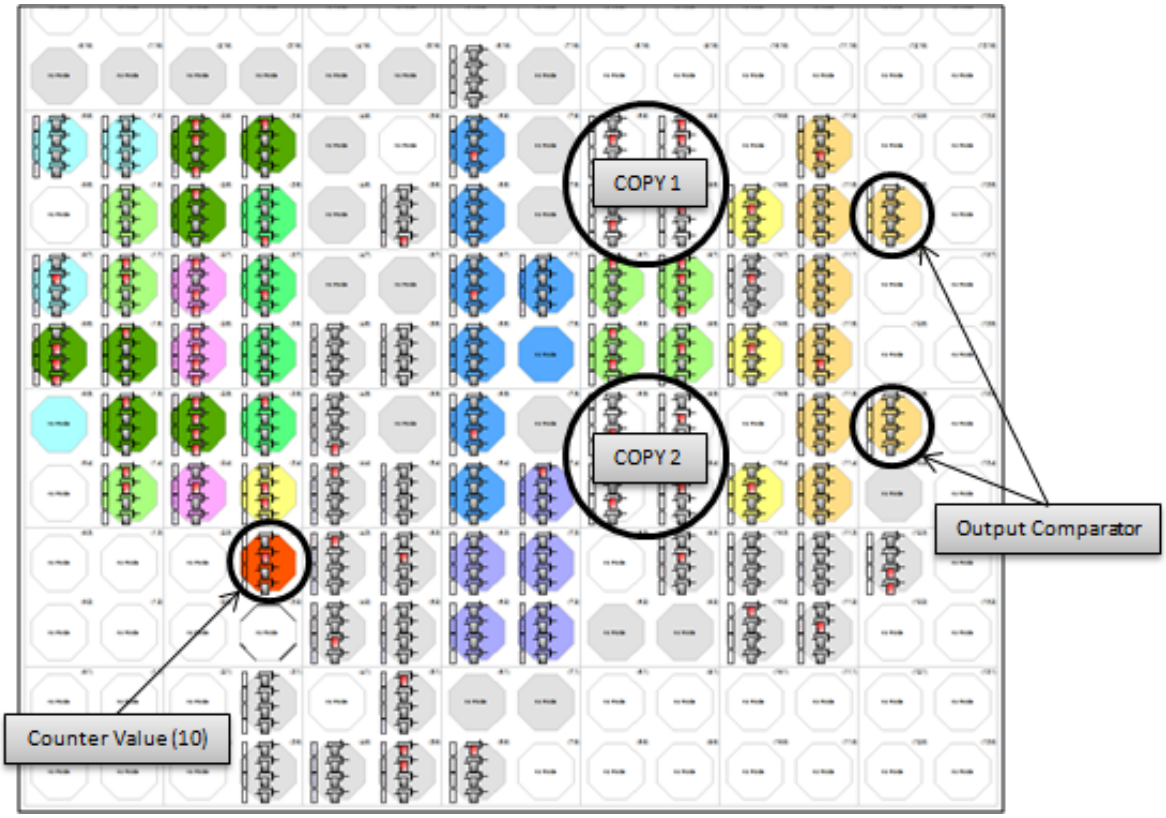

Figure 4.11: Simulation View. 2FU mode

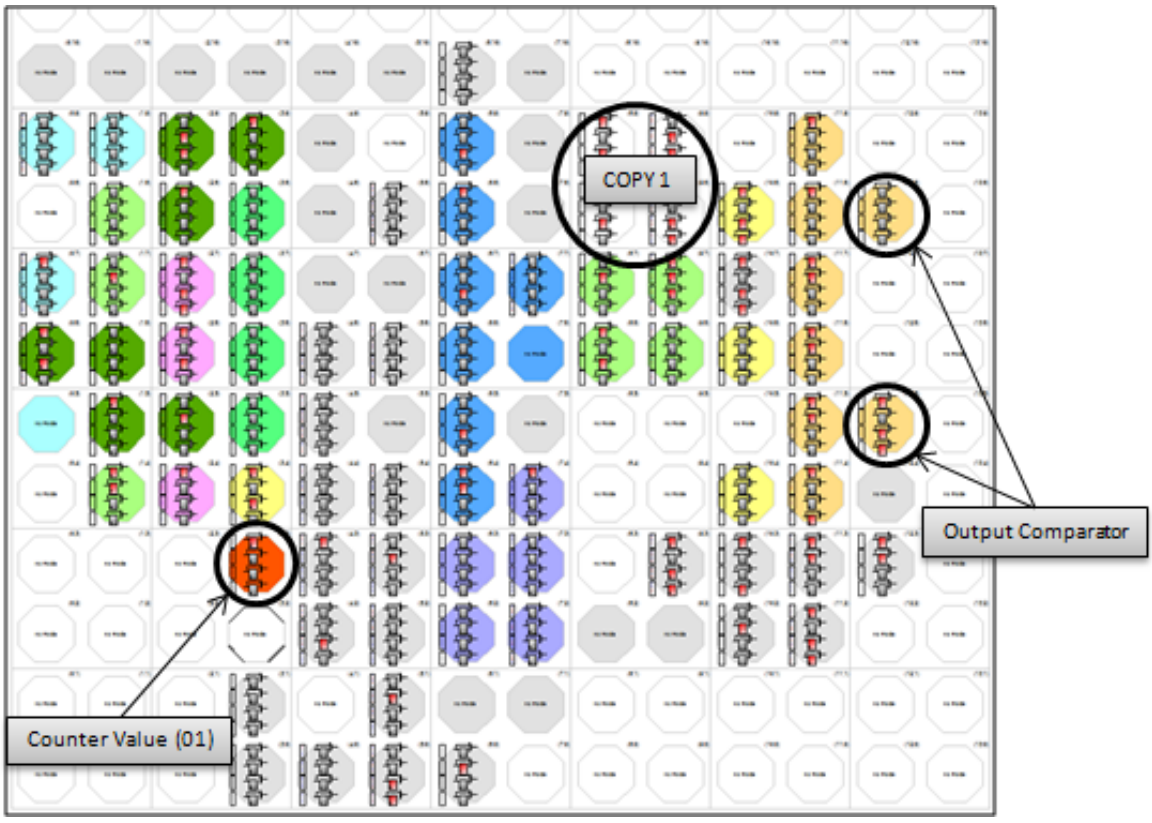

Figure 4.12: Simulation View. 1FU mode

5 VHDL Implementation

Several limitations on Ubichip framework, mainly due to the issues discussed in the previous chapter kept the author from implementing the full system initially designed as discussed in chapter 3. One of the recommendation the author has for future of Ubichip framework is to implement dedicated hardware for the fault tolerant system outside the Ubicell array, which would not only make the fault tolerant system more reliable but also make the framework ready to be used for more practical application where reliability is important.

This chapter will explain how the author has developed a fault tolerant system using a synthesizable VHDL code. The working of the system has been tested and verified on simulation.

5.1 Overall Design

The VHDL implementation realized a state shown in the Figure 3.8 in the earlier chapter. Figure 5.1 shows the block diagram of the system. The goal of this VHDL design was to have all the framework in Power Aware Fault Tolerant System to be implemented as dedicated hardware so that they do not have to take up the Ubicell space. To achieve this purpose, blocks consisting the SR controllers as well as the FU spaces are made to operate exactly as they do in the Ubichip. Voters for the Triple Modular Redundancy (TMR), Fault Locater, and the control FSMs are designed with synthesizable VHDL code.

Control FSM:

In normal operation, the FSM controls the number of FU copies by consulting the power mode, which is calculated externally depending on the power consumption of the system.

The FSM detects system fault when the signal from the fault locater indicates that one of the Functional Units (FU) is faulty. When an error is detected, the configuration of the FU is cleared and the system resets to re-configure the FU spaces. The FSM has internal counter to keep track on how many times errors are detected on each FU, when the error count exceed the predefined number (1 in this experiment), the FU is marked as invalid, and the system discards the FU space.

Ubichip Emulation:

The Functional units, and SR controllers in this implementation emulates exactly the hardware of Ubichip. The control bus for the SR controller and the Ubicell, as well as Ubicell with four 4-bit LUTs are realized to test operation of the system.

Fault Tolerance Circuit:

Since the system is power aware and the number of FU copies differ at different power mode, the FSM notifies the power mode to the fault tolerant circuit for them to behave according to available resources. When 2 FU copies are available, making a full TMR possible, the voters vote on the output from three FUs, and the fault locater detect the occurrence of error and its location. When only 1 or 2 FUs are available, the voter simply passes the output from the original FU. During SR processes and when error is detected, voter outputs is kept to logic '0'. When there are 2 copies of FU (fault detection mode), the locater detects errors by comparing the output from the 2 FUs.

Operation Mode:	Voter Output
1 FU	Original FU
2 FU	Original FU
3 FU	Vote result
Others	logic '0'

Table 5.1: Voter Outputs and Operation mode

5.1.1 Functional Test

All the possible combination of state transitions in normal operation shown in the Table 5.2 were verified as well as the fault tolerant operations shown in the talbe 5.3.

Previous State:		New State:
1 FU	$\rightarrow$	2 FU
1 FU	$\rightarrow$	3 FU
2 FU	$\rightarrow$	1 FU
2 FU	$\rightarrow$	3 FU
3 FU	$\rightarrow$	1 FU
3 FU	$\rightarrow$	2 FU

Table 5.2: State Transitions Verified

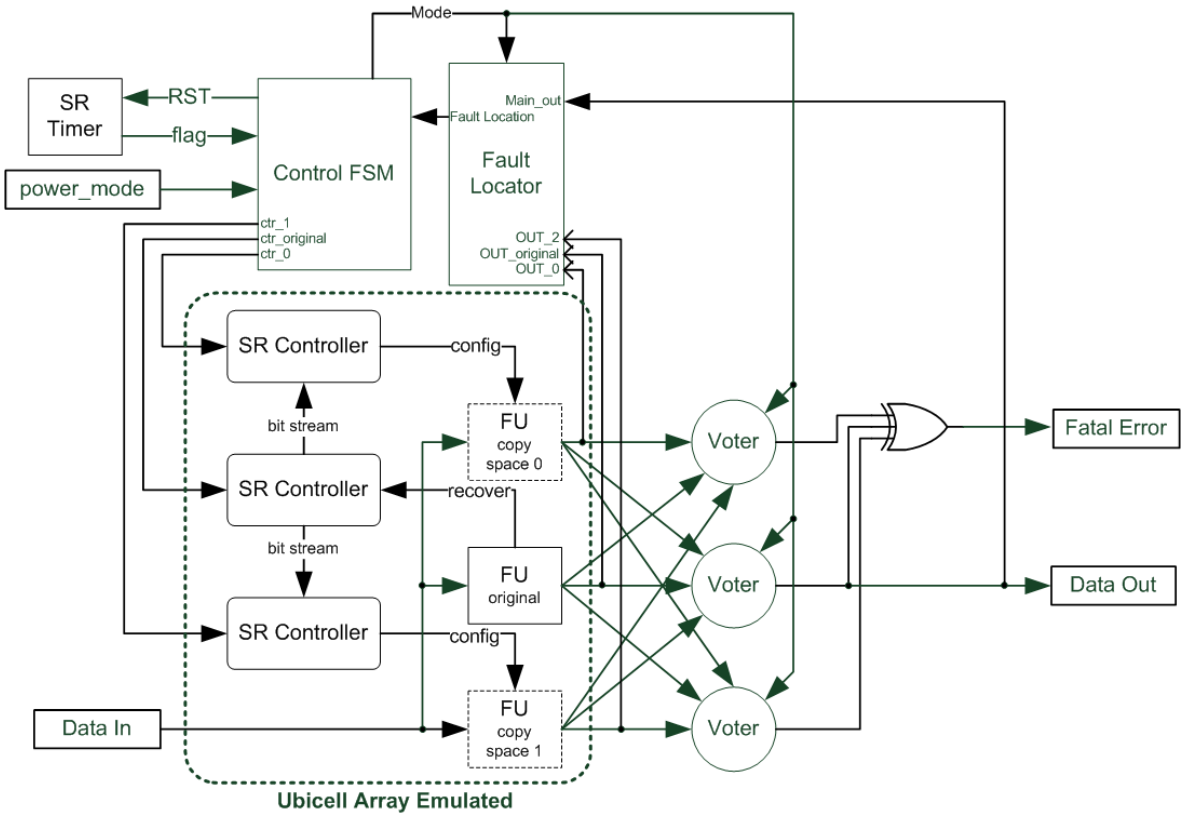

Figure 5.1: Block Diagram of the VHDL Implementation

State:	Error:	Action:
2 FU: Normal	Error Detect	Reconfigure
2 FU: FU1 invalid	Error Detect	Reconfigure
3 FU	FU0 error	Reconfigure
3 FU	FU0 error 2nd	FU0 Invalidate
3 FU	FU1 error	Reconfigure
3 FU	FU1 error 2nd	FU1 Invalidate
3 FU	FU2 error	Reconfigure
3 FU	FU2 error 2nd	FU2 Invalidate

Table 5.3: Verified Fault Tolerance Functions

6 Conclusion

6.1 Analysis

6.1.1 Ubichip Implementation

6.1.1.1 Cell Count, Area Overhead

Table 6.1 shows the number of cells used for each system block. The fault tolerant system takes a total of 64 Ubicells. The area overhead in this application is significant because a very primitive 4-cell single MC was used as the FU. When a larger FU is implemented, the area of the rest of the system remains unchanged. As the size of Ubicell array in Ubichip is 10 by 10 MCs (20x20=400 Ubicells), the area overhead of this fault tolerant system is 16% of the total array area.

Block:	# of cells:
Control FSM	22
SR Controller (x3)	12
Output Comparator	11
SR Timer	5
Transition Counter	14
Functional Unit	4

Table 6.1: Cell Count of the Design

6.1.1.2 Timing observation

Configuring cells using a serial register means that the time required for configuration increases as the number of MCs to be replicated increases. Table 6.2 shows the cycles required for the SR unit to complete for different number of MCs. The worst-case estimation for the operating frequency of Ubichip is 50MHz. Since the operation of the FU must pause during the replication process, the replication time especially for larger FUs may become a serious issue for timing-critical applications.

Number of MCs:	Clock Cycles:	Time in seconds:
1	547	10.9μs
2	1,072	21.4μs
10	5,272	105μs
300	157,522	3.15ms

Table 6.2: Clock Cycle and time in seconds required for Self-Replication (at 50MHz)

6.1.2 Issues and Advantages

Based on the experience of the design and implementation process, the author has identified the advantages as well as issues in using Ubichip.

Advantages:

The biggest advantage in using Ubichip for this application is the built-in Self Replication (SR) capability. It allows configuring logic circuit of any size or shape to be configured on any part of Ubichip dynamically. Furthermore, because Ubichip as a bio-inspired architecture is based on an array of Ubicells, the configuration of a new logic circuit can be done locally instead of discarding the entire column as in the case of FPGAs. As the name suggests, the Self Replication process is completely autonomous; multiple SR processes can occur simultaneously on single array space. This allows replication of several different circuits, or making multiple copies of a single circuit. Furthermore, by incorporating output values in the configuration stream, state is preserved when copies of a circuit is made. Furthremore, the Ubichip is designed in a way that signals can reach any part of the array in single clock without buffer so that designers can implement any logic circuit without worrying about the critical path length.

Issues:

- The major problem of implementing a design on Ubichip is the lack of any EDA tools. This is inevitable as Ubichip is an academic prototype platform. However, lack of EDA tools makes implementation of complicated system nearly impossible
- SR is a versatile and useful function of Ubichip. However the configuration bit stream is sent through serial chain of registers, making the time it takes for the replication process to be significant
- Implementing the FSM and control circuit on Ubichip platform have been proven to be error prone and time consuming. Dedicated hardware that includes SR controller and timer can address this issue. In addition, a standard FPGA or Microcontroller (MCU) circuit to implement general controller of the system could also allow more conventional use of Ubichip. Furthermore, Standard CMOS logic circuits, compared to SRAM based FPAG implementation are more tolerant to radiation; reducing the possibility of system malfunction due to fault in the fault tolerant circuit itself

6.2 Recommendation / Future Work

Implementation on Ubichip:

The author was not able to have access to fabricated Ubichip in time. Once the chip is available the design should be loaded and tested on Ubichip for further verification.

Power Measurement:

In this project for the Ubichip implementation, number of logical transition of the output signal was counted to estimate the system power consumption. Needless to say this method cannot measure the accurate value. The author has proposed a simple analog circuit based on Built-in Current Test (BIC) to measure the system current consumption. The design should be considered for integration for better perform as a power aware system.

Dedicated Controller Block:

Although having flexible capability, Self Replication Unit of Ubicell require control circuit, which must be implemented in Ubicell array. A System controller, which includes FSM and SR controller could be implemented externally so that the Ubicell array can be used solely for the functional unit.

SR Stop Flag:

H-Flag, which indicates the shape and size of organelles during SR process contain a stop flag. This flag should be interpreted by the SRCU to stop the process so that SR timer is no longer necessary.

Advanced Fault Tolerance:

The system developed in this project can be modified to implement more advanced fault tolerant functionality. On Ubichip platform, the author recommends next step to be the ability to discard the original cell when fault is found there. Furthermore, area of reconfigurable fabric should be isolated as permanently faulty after certain number of faults are found at the same cell.

Development of the IDE Environment:

Problems the author experienced due to the lack of EDA tool was mentioned earlier as one of the issues. Here the author lists types of functions and tools that can significantly increase the ease of design implemntation on Ubichip:

- **Routing tool:** During the implementation, the routing of the signals was the most time-consuming stage as few each signal needed to be routed by configuring the I/O selection of all the Ubicells that signals pass through. As output of each Ubicell is only 4-bits, routing also required well-thought floor planning of the Ubicells. A routing tool, which can convert a list of signals to a map of Ubicells with routing configuration should be implementable, and this can greatly improve the implementation process

- **Netlist Generation:** The ability for the designer to name the signals and view them visually in an organized way can dramatically enhance the effort and time required for debugging
- **Netlist Highlighting:** If GUI can represent each signal on the routing view, design process as well as debugging process would be easier for the designers
- **Wave Viewer:** Ubimanager incorporates the Modelsim EDA tool for Ubichip simulation. It would make the debugging effort simpler if the wave viewer function of the Modelsim is accessible from Ubimanager to trace the signals during simulation
- **High-level Language Support:** Lack of high-level language support makes Ubichip a unpractical option for many designs. While it is not simple task to develop a compiler that can convert VHDL or System C into Ubicell circuit, standard FPGA fabric maybe added in addition to implement some of the control circuit.

Use of ALU mode:

Although a Ubicell contains a 4-bit ALU, Currently ALU mode of the Ubicells is not available for logic circuit design as they are designed only for SIMD mode of Ubichip. Availability of ALU can expand the possibility of more complicated systems to be implemented on Ubicell array

Expanded Signal Connectivity:

The output data width of Ubicell should be expanded to allow more flexibility in signal routing. Furthermore, an additional general purpose data bus used for essential signals such as reset and FSM state can simplify the logic circuit implementation

6.3 Concluding Remark

The motivation of this project was to address the serious issues facing VLSI design namely the power consumption and reliability. Power consumption is a bigger issue when feature size of IC becomes smaller. Radiation induced soft errors, also called Single Event Upset (SEU) is a serious reliability concern for modern VLSIs especially for reconfigurable circuit as SEU occur at higher rate on SRAM blocks, which are the bases of most reconfigurable circuits.

Ubichip, a bio-inspired reconfigurable hardware developed in Perplexus project is constructed with an array of Ubicells, which are the reconfigurable blocks capable of Self Replication (SR). The SR capability seemed especially promising for implementing Triple Modular Redundancy (TMR) based fault tolerant system. SR units' capability of creating and destroying the copies of circuits dynamically can also be used to control the size of system in order to regulate the overall power consumption.

With the capability of Ubichip in mind, a power aware fault tolerant system was designed, which is then implemented on Ubichip platform. During the implementation, several limitations and issues of Ubichip platform were identified; some of the limitations kept the author from implementing complete system. Nevertheless a proof of concept design was successfully implemented and functions were verified. Identified issues were listed in this report, which leads to some of the recommendations for the future research area.

The design implementation on Ubichip was also reported in a paper *Implementation of a Power-Aware Dynamic Fault Tolerant Mechanism on the Ubichip Platform*, which was accepted for 2010 International Conference on Evolvable Systems (ICES).

Appendix A

A.1 Ubichip LUT Configurations

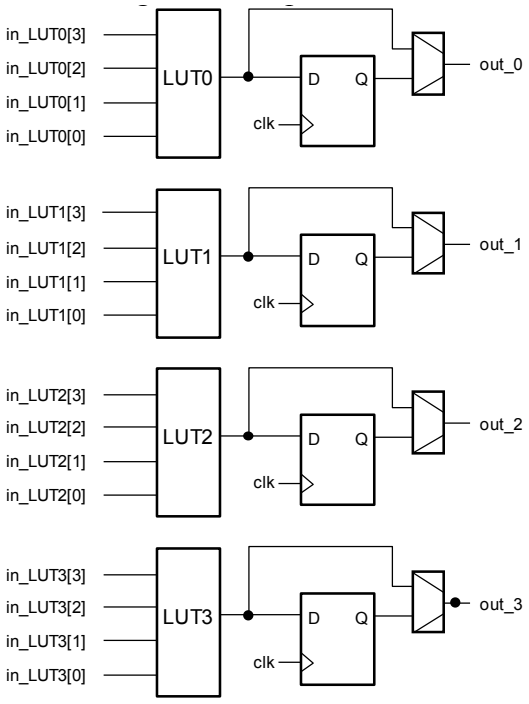

Figure A.1: Ubicell Mode: 4 Independent 4-input LUT

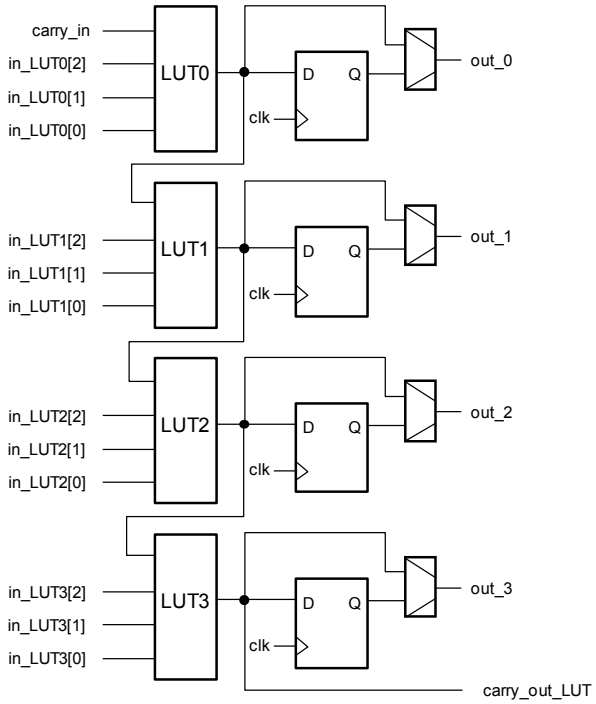

Figure A.2: Ubicell Mode: Wide Decoder/high-fanin

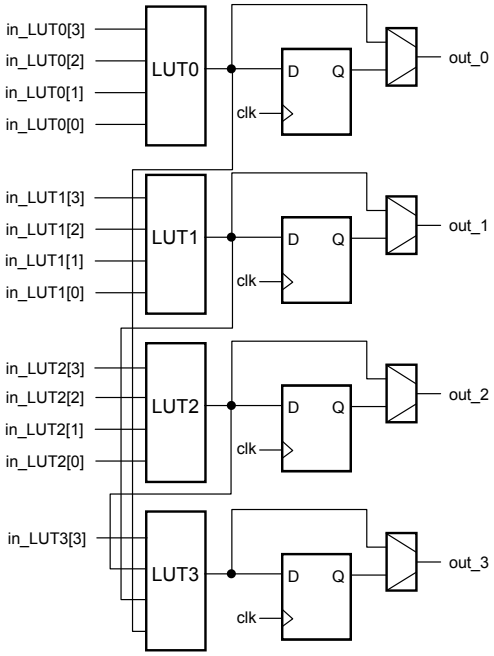

Figure A.3: Ubicell Mode: 2-Level Logic

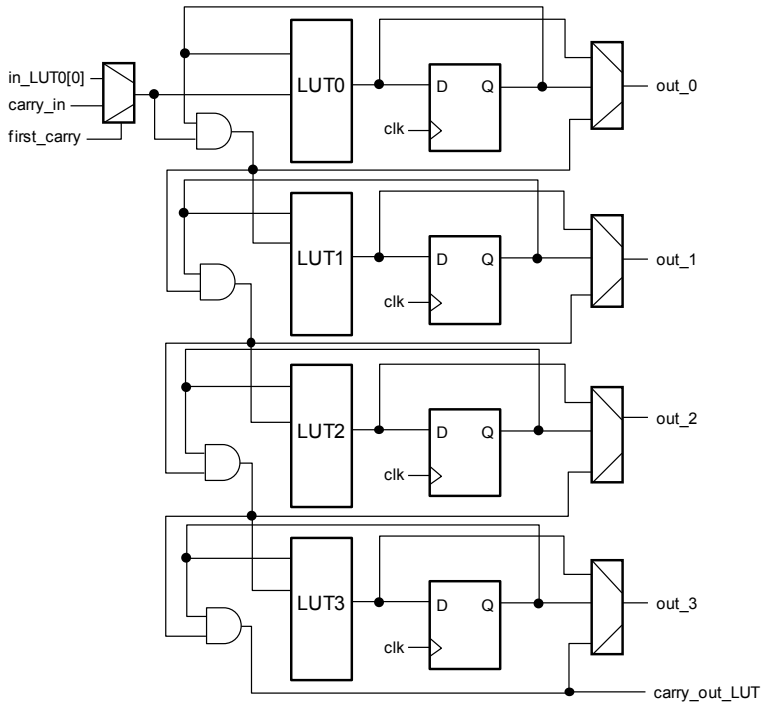

Figure A.4: Ubicell Mode: Counter Mode

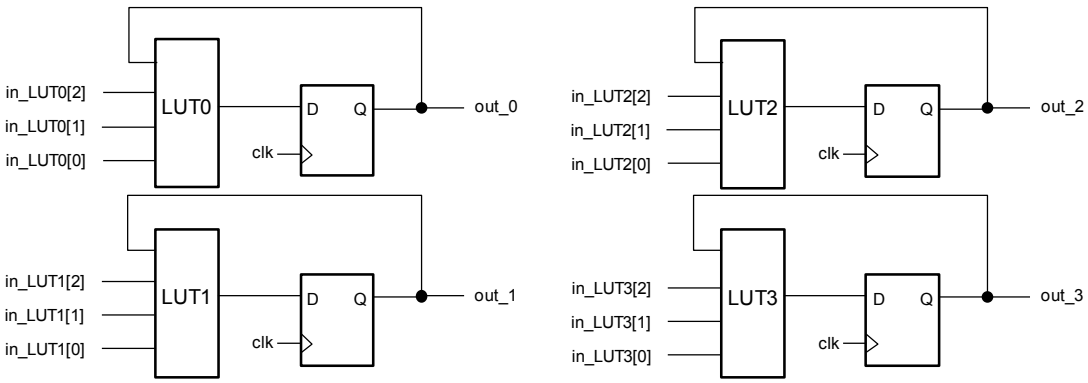

Figure A.5: Ubicell Mode: 1bit State Machine

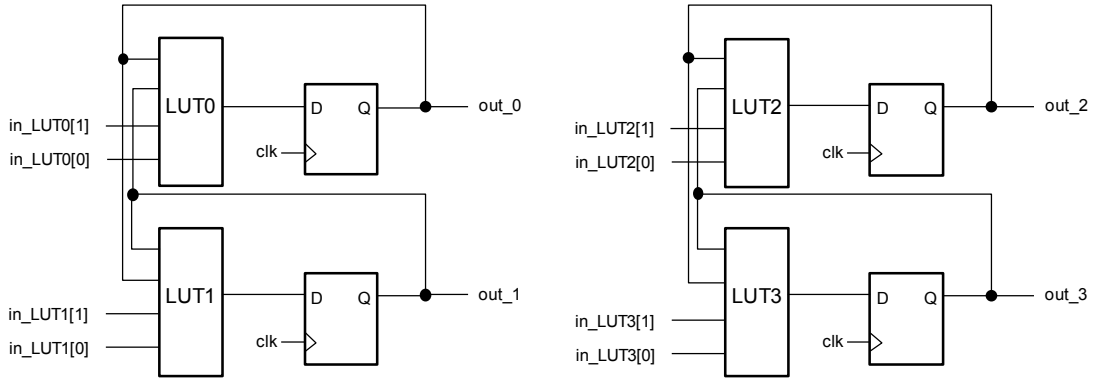

Figure A.6: Ubicell Mode: 2-bit State Machine

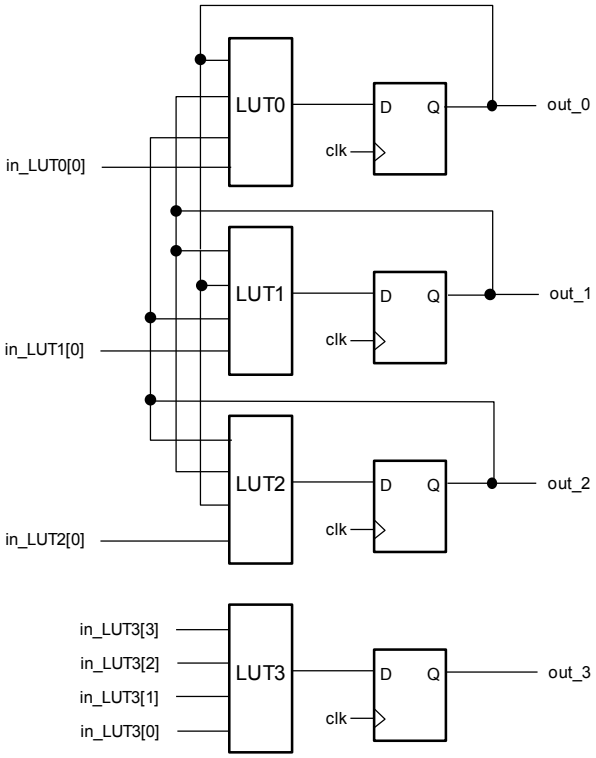

Figure A.7: Ubicell Mode: 3-bit State Machine

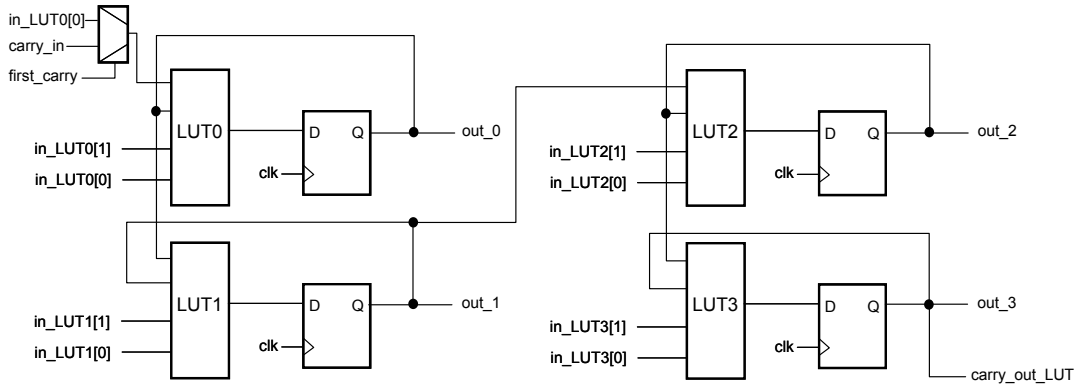

Figure A.8: Ubicell Mode: Shift Register

A.2 Schematics

Figure A.9: FSM State Diagram designed in HDL Designer software Package

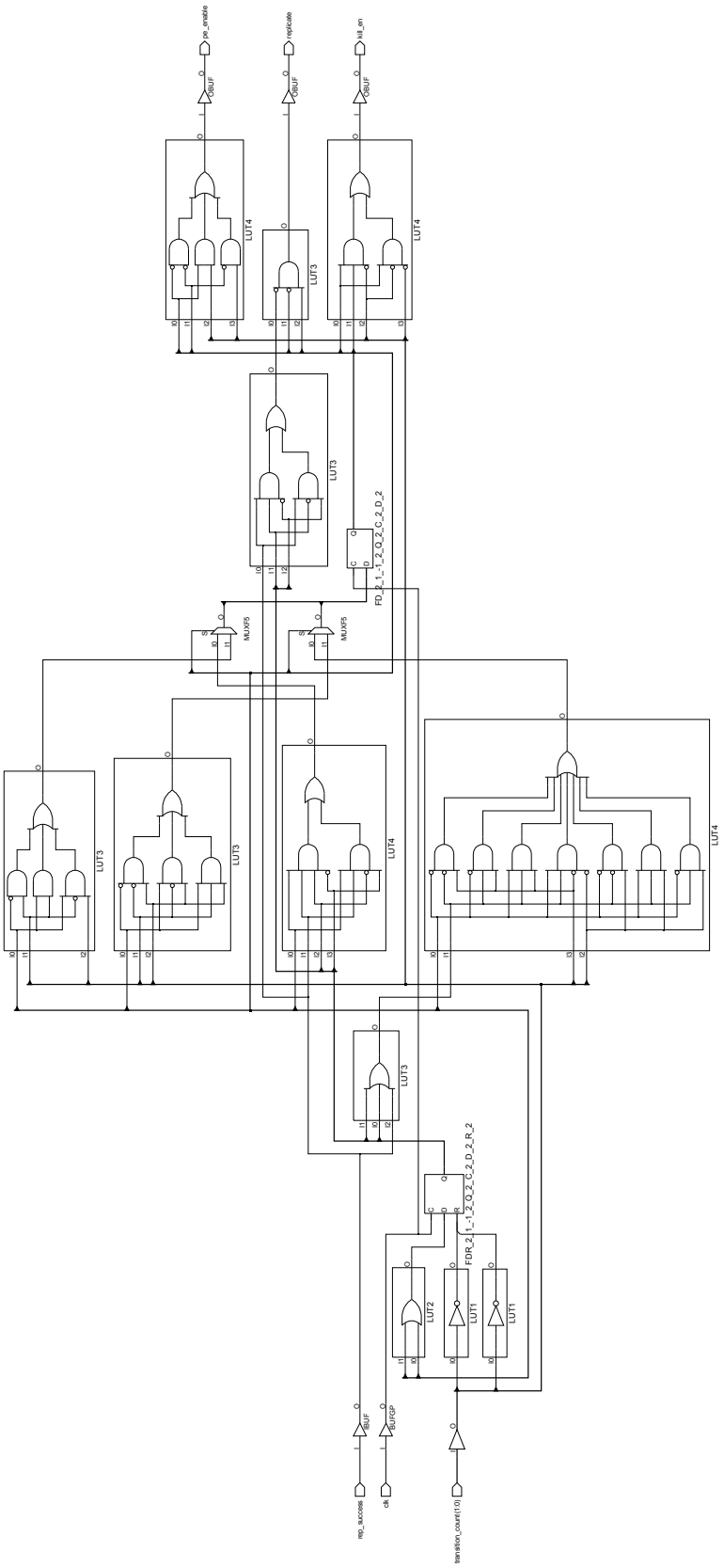

Figure A.10: LUT Schematics Generated by Precision RTL Synthesis

A.3 Dynamic Routing simulation

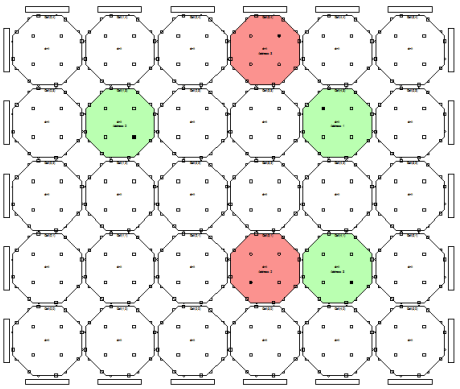

Figure A.11: DR Simulation: Before the Process

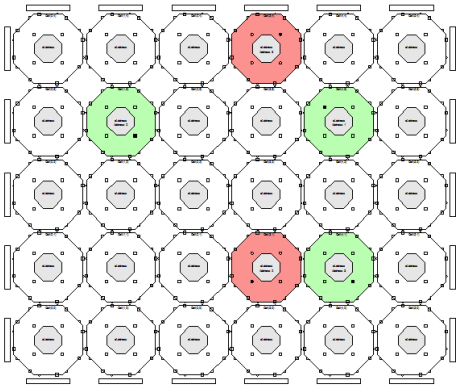

Figure A.12: DR Simulation: Master Search

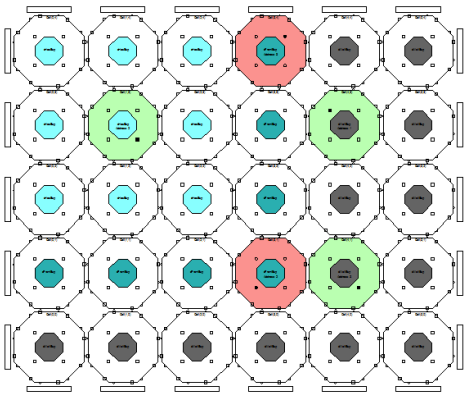

Figure A.13: DR Simulation: Expansion

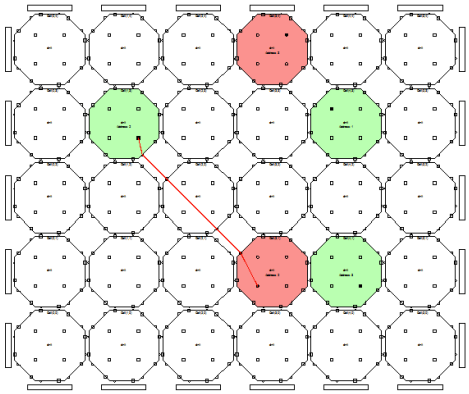

Figure A.14: DR Simulation: Path Created

A.4 Ubimanager GUI Screen Shots

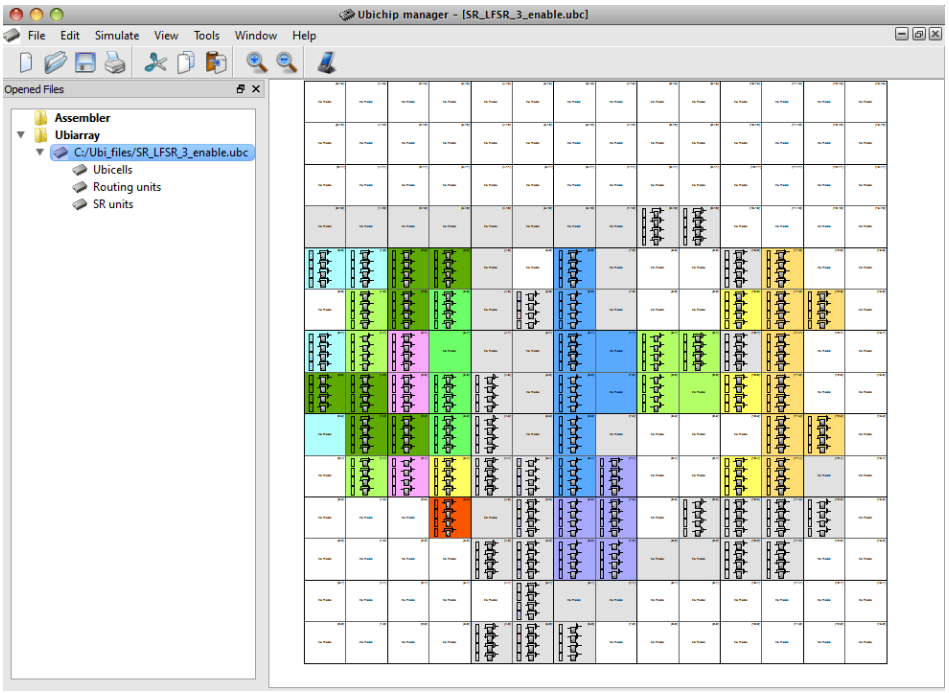

Figure A.15: Ubimanager: Standard Screen with Ubicell Array

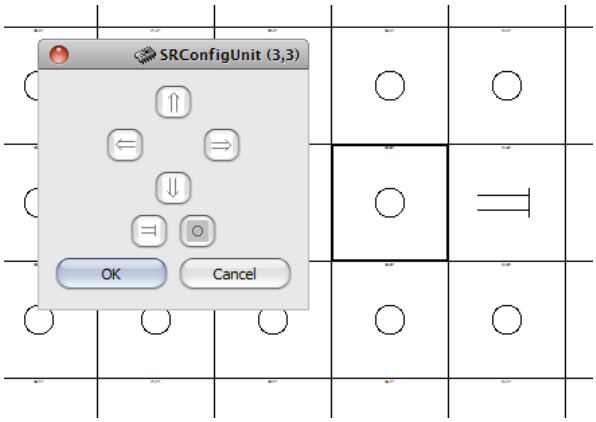

Figure A.16: Ubimanager: Self Replication (SR) H-Flag Configuration

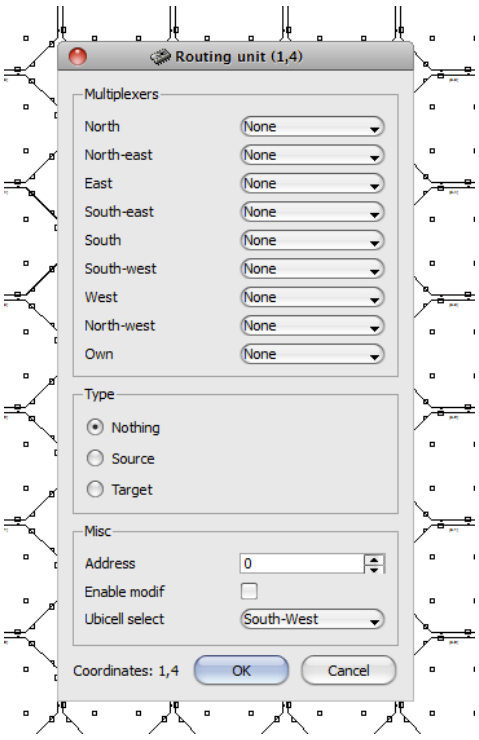

Figure A.17: Ubimanager: Dynamic Routing (DR) Routing Unit (RU) Configuration Window

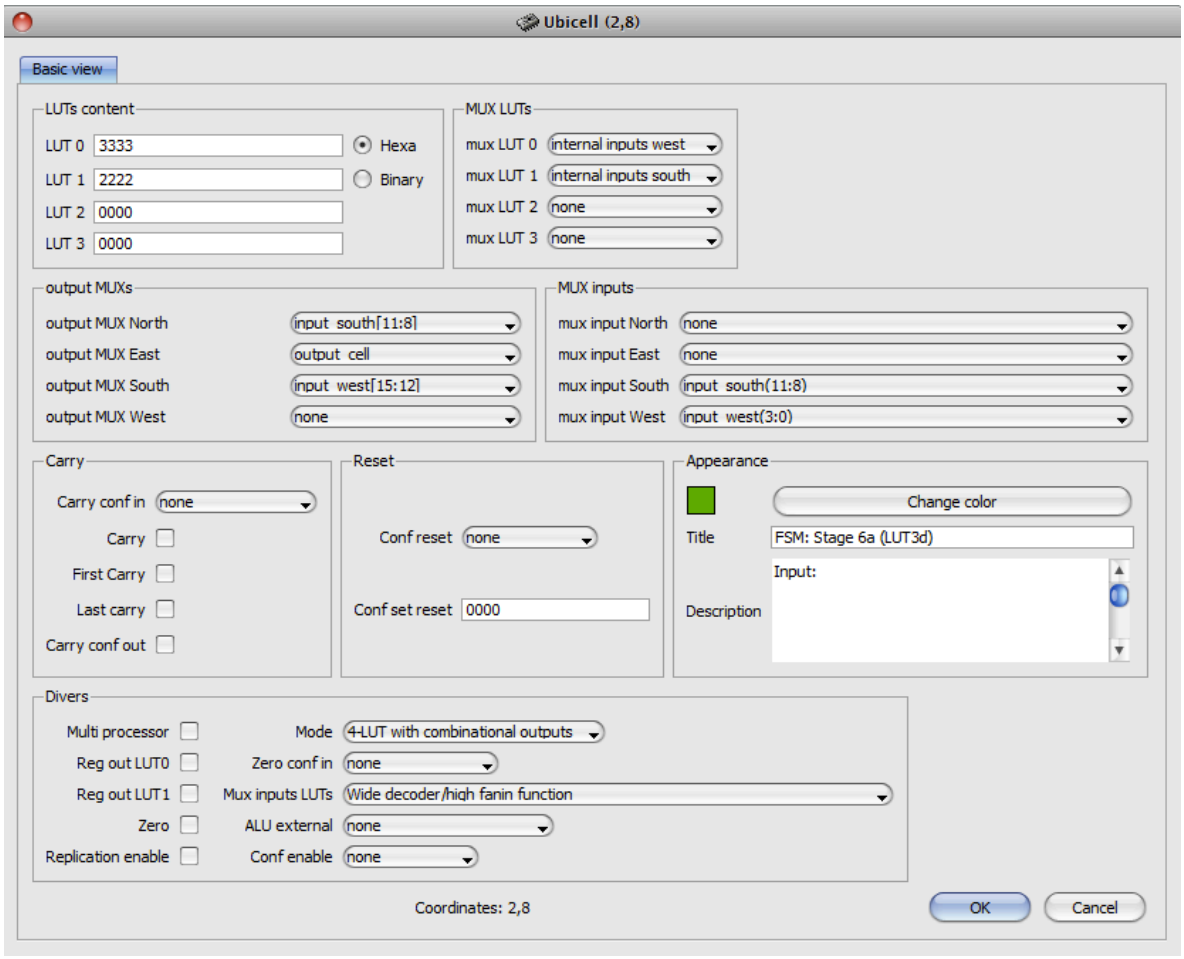

Figure A.18: Ubimanager: Ubicell Configuration Window

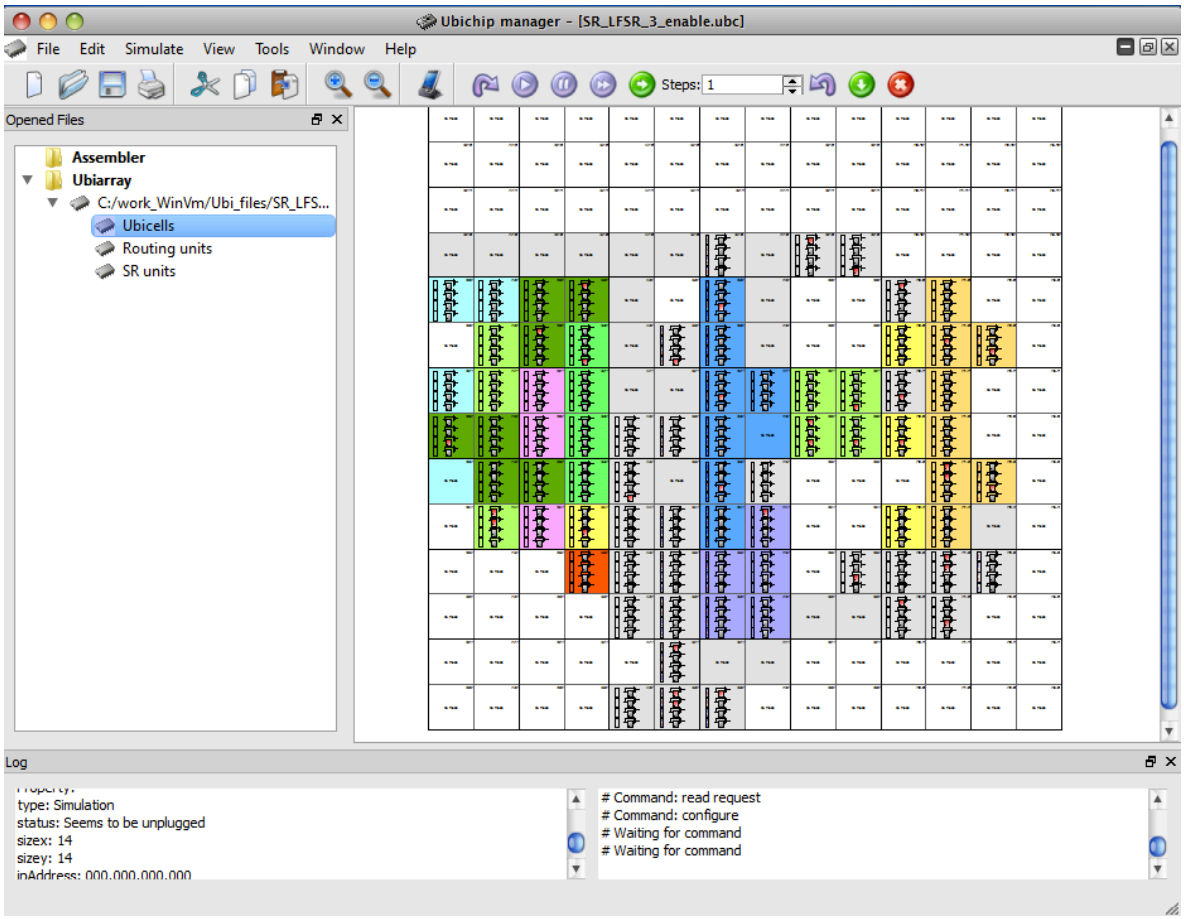

Figure A.19: Ubimanager: Simulation Window

A.5 Paper Accepted for 2010 International Conference on Evolvable Systems (ICES)

Implementation of a Power-Aware Dynamic Fault Tolerant Mechanism on the Ubichip Platform

Kotaro Kobayashi[1], Juan Manuel Moreno[2], Jordi Madrenas[2]
k.kobayashi-1@student.tudelft.nl
moreno@eel.upc.edu
madrenas@eel.upc.edu

[1]Delft University of Technology, Delft, The Netherlands
[2]Universitat Politècnica de Catalunya, Barcelona, Spain

Abstract. Dynamic fault-tolerant techniques such as Built-in Self Repair (BISR) are becoming increasingly important as new challenges emerge in deep-submicron era. A dynamic fault-tolerant system was implemented on the Ubichip platform developed in the PERPLEXUS European project, which is a bio-inspired custom reconfigurable VLSI. The system is power-aware; power consumption is monitored dynamically to regulate the number of copies made by a self-replication mechanism. This paper reports the design, implementation, and simulation of the fault-tolerant system.

Keywords: Dynamic Fault Tolerance, Self-replication, Reconfiguration, BISR, Bio-inspiration, Ubichip, PERPLEXUS, Power-awareness

1 Introduction

The IC technology scaling, which follows the famous Moore's law has evoked a great deal of advancement in modern electronics for the last few decades. Designers have been able to integrate greater number of transistors on a limited area of silicon die; modern VLSI systems with multiple function blocks on a single die allow designers to reduce the physical size of the systems and manufacturing costs. The ITRS predicts in [2] that the gate length of VLSI systems will go below 20 nm in the later half of this decade, a length enough to fit only few hundreds of silicon atoms in one line. This deep-submicron paradigm poses new challenges to the VLSI design; intricacy of the fabrication will be greater, so that manufacturing defects will likely increase while testing for those defects will be very challenging due to the ever increasing complexity of the system. The reliability will also suffer due to phenomena such as gate insulator tunneling, Joule heating, and electromigration. Furthermore, the small feature size will certainly increase the unpredictable errors due to alpha particles, namely soft error, or Single Event Upset (SEU) [1], [2] .
There have been many advancements in techniques such as Design for Test (DFT) and Built-in Self-test (BIST) [1]. While these tests can effectively detect faults due to defects, they cannot detect unforeseeable faults caused by

aging-defects or temporal faults such as SEU. In order to assure the reliability while incorporating deep-submicron technologies, the system should have dynamic fault-tolerance capabilities to detect and correct errors on the run. If a VLSI system can autonomously detect and correct an error situation dynamically, it will not only increase the reliability but also the yield and life-time of the ICs, resulting in a significant cost reduction [5].
The Ubichip is a bio-inspired custom reconfigurable VLSI system developed in the PERPLEXUS project [6], [10]. Ubichip offers bio-inspired capabilities such as dynamic routing and self-replication. The operational flexibility provided by these mechanisms makes Ubichip an ideal platform to implement dynamic fault tolerant systems with Built-in Self Repair (BISR) capabilities.
This paper presents the design, development, and simulation of a power-aware fault-tolerant system implemented on the Ubichip. Section 2 discusses the background and overall system architecture. Section 3 briefly introduces the Ubichip platform used in this experiment. Section 4 describes the implementation of the design in detail. Section 5 discusses the implementation and simulation results. Finally, the future research areas as well as concluding remarks are included in section 6.

2 A Power-Aware Fault Tolerant System

2.1 Background

In order to protect a system from logic errors during run-time,it can use Built in Self Repair (BISR). Several different methods of implementing BISR are discussed in [5]. Triple Modular Redundancy (TMR) is a widely known method of BISR. Although it is also known to be area consuming, it is very simple to design and unlike error correcting codes [3], no static specialized design tools are required; it is more versatile in accommodating different logic circuits.
In dynamic reconfigurable systems, Funcion Units (FU) are configured at run-time as required. Unused FUs can simply be deleted to give more space for necessary functions. In such systems, the same TMR circuit can work for different FUs configured in the same area because of the simplicity of the algorithm.

2.2 Power Awareness

The power consumption must be considered when implementing TMR. Having three identical circuits would result in at least three times more power consumption in terms of switching current. Furthermore, power consumption is a major issue to be solved in VLSI today; larger circuits, higher operation frequency, and smaller feature size all contribute to higher power consumption.
TMR is intrinsically not a power efficient design technique. In order to reduce the effect on power consumption, authors have implemented the power-aware TMR system based on a previous work presented in [11]; the system monitors

its power consumption and eliminates one or both of the TMR copies when the power consumption is above a predefined threshold.
While clock gating or power gating also can be used to control the power consumption of TMR designs in the same way, our framework on Ubichip is capable of dynamic reconfiguration, thus same FU space can be used for different blocks according to power consumption and operation phases.

2.3 System Description

Figure 1 shows the FSM states of the power aware design presented in this paper. Initially the system starts with a single functional unit (FU). As our system platform (Ubichip) does not have current sensing capabilities, the power consumption of the running application is measured by a 'transition counter'. This subsystem estimates the power consumption by means of a 'counter value' and controls the number of FU copies using the self-replication (SR) function of the Ubichip. Counter value is computed by accumulating output values from multiple clock cycles and counting the number of transitions. When the number of transitions from the original FU is the highest, meaning in this case more than 3 bits transition in 2 consecutive clocks, the counter value is '00' and no copies of FU are made. When the number of transitions is low, meaning the transition from the original FU is between 0 and 1 bit for 2 consecutive clocks, the counter value becomes '10', which leads the system to create 2 copies of FU. Counter value '01' is an intermediate transition count; when the output from the original FU has 2-bit transition for more than 2 clock periods, only one copy of the FU is created.
The system starts with single FU mode. After few clock cycles the transition counter estimates the current consumption and indicates it as 'counter value'. The system constantly monitors its power consumption and changes the number of FU copies accordingly.

3 A Reconfigurable Framework: PERPLEXUS

The system was designed within a framework developed in the PERPLEXUS European project. The Ubichip is the kernel of this project; a reconfigurable VLSI system endowed with bio-inspired capabilities. Details of the PERPLEXUS project can be found in [10], [6].

3.1 Ubichip

Ubichips are mounted on a prototype system called Ubidule, explained in [10]. A Ubichip consists of three major blocks: An array of reconfigurable processing elements called Macrocell (MC), the System Manager and a controller for Content Addressable Memory (CAM). The system manager block is responsible for configuring the reconfigurable array and external communication. Each MC is made up with four reconfigurable cells called Ubicell, which is explained

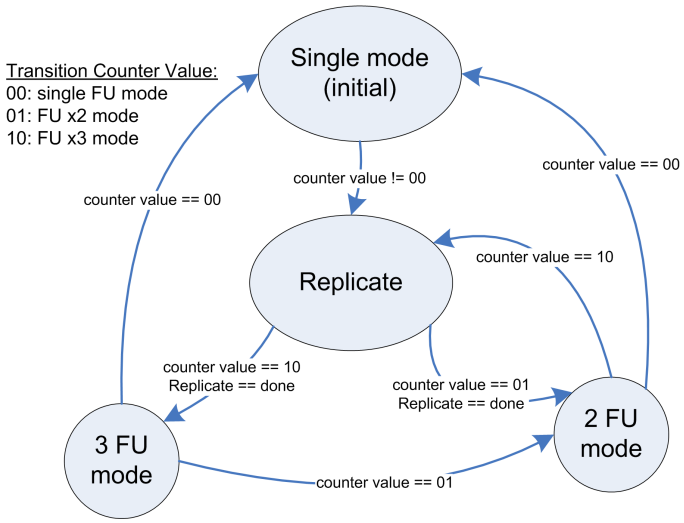

Fig. 1. FSM State Diagram of the implemented system

later in this section. The configuration bit stream for each MC can be recovered and configured dynamically using the Self-Replication (SR) function of the Ubichip. The SR function is used extensively in this project, thus its details are briefly explained later in this section. Each MC also contains a Dynamic Routing (DR) control unit, which allows a pair of MCs to establish communication paths dynamically. The DR functionality of Ubichip is further explained in [7]. Furthermore, a Ubichip can also be configured in multiprocessor mode where a SIMD-like parallel machine can be implemented.

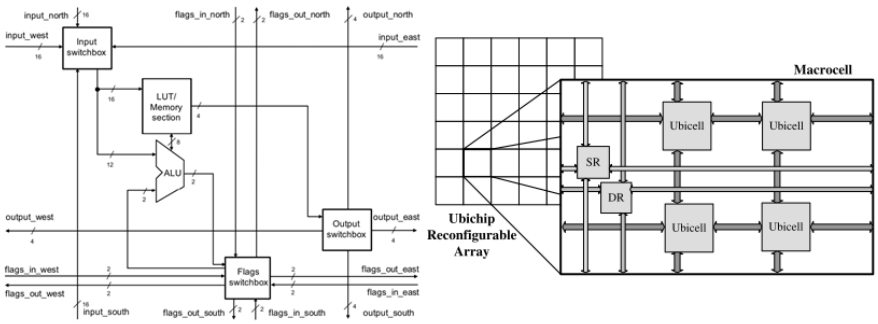

Fig. 2. Organizatio of a Ubicell (Left), Ubicell array and Macrocell (Right)

3.2 The Ubicell

Figure 2 shows the overall organization of a Ubicell. As explained extensively in [4], a Ubicell can be configured to implement various logic functions in LUT mode or work as a part of multi-processor machine in ALU mode. In this project all the cells are configured to various configurations within LUT mode.

3.3 Inter-cell Connection

Neighboring Ubicells can be connected by selecting appropriate input/output multiplexers. Figure 3 shows the neighborhood connectivity among Ubicells. The output multiplexers are able to choose not only the output but raw input from other neighbor cells as well. Furthermore, it is possible for any pair of macrocells (4 Ubicells) to communicate using the Dynamic Routing (DR) capability.

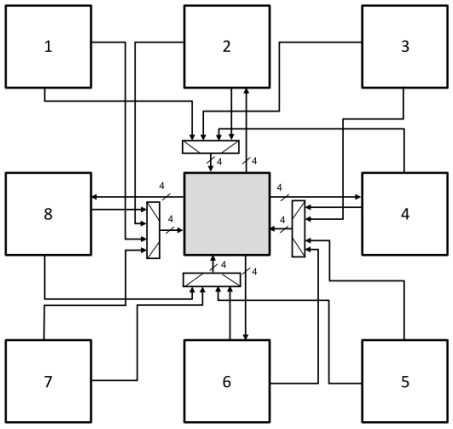

Fig. 3. Inter-Ubicell Connectivity

3.4 Self Reconfigulation

A group of more than one macrocells (organism) can be copied to other parts of the Ubicell array using the Self-Replication (SR) mechanism. An organism has the configuration bits of its MCs connected by a chain of shift registers. The configuration bits of MCs can be recovered through this chain by a SR controller. The SR controller can use this recovered bit-stream to configure an empty area during self-replication process. Details of the SR controller on Ubichip are explained in [9].

3.5 Ubimanager

The authors used a software tool called Ubimanager, which was designed in the PERPLEXUS project in order to manage the Ubichips. The Ubimanager allows developers to design Ubichip implementations by means of a GUI environment; developers can configure all the three layers of Ubichip: Ubicells, Dynamic Routing Units (DR), and Self-Replication Units (SR). It is also capable of simulating the implementation using Modelsim. A detailed description of Ubimanager tool is provided in [8].
In a Ubimanager environment, the array of Ubicells is represented in a GUI window; a developer can configure each cell by double-clicking the cell to open the configuration window.

4 Implementation

Figure 4 shows shows a block diagram of the dynamic fault tolerant system implemented on a Ubichip. There are three SR controllers; one is responsible for reading the configuration bit-stream from the original FU, being the other two responsible for replicating the copies. The control signals for the SR controllers are created in the 'Control FSM' block. The level of system power consumption is sent to the control FSM from the 'Transition Counter' block as 'counter value'. According to the power consumption level the FSM changes the operation mode and forces the SR controller to have an appropriate number of FU copies. Every time new copies of the FU are made, the Control FSM block relies on the signal from the 'SR Timer' block to stop the SR process upon completion. The outputs from FUs are compared at the 'Output Comparator' block.

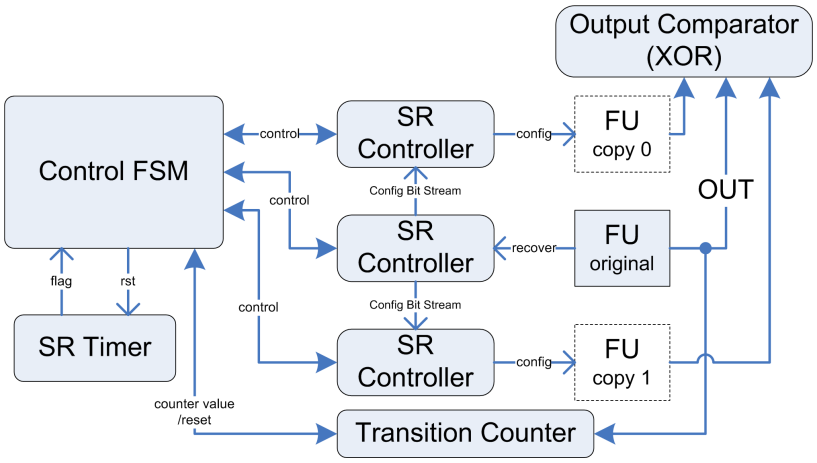

Fig. 4. Power Aware Fault Tolerant System: Overall Block Diagram

The functionality of its main building blocks is the following.

Functional Unit (FU): As the goal of this experiment is to show a proof of concept working system, the implementation of the Functional Unit (FU) was kept simple; a combination of memory, counter and pseudo-random number generator (LFSR) was configured in a MC as shown in figure 5.

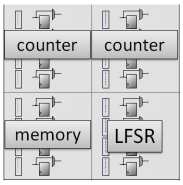

Fig. 5. Functional Unit (FU)

Output Comparator: In this implementation, output comparator simply indicates the bit-wise XOR of the outputs from each FU. In a future implementation, the comparator result should be fed back to the controller to implement error correction.

SR Controller: While it is possible for one SR controller to remove and make a copy of an organism (set of MCs), 'remote configuration' explained in [9] is necessary to control two different copies separately. In this case, a total of 3 SR units are required: one for recovering the configuration bit stream of the original organism, one each for configuration of the two copies.
The SR mechanism of Ubichip blocks the output from the MCs during the SR process, eliminating the need to filter erroneous output during the SR process. Furthermore, values of each register in the MCs are incorporated in the configuration bit stream; states of the circuits are preserved to the newly created copy of an organism.

SR Timer: A 4-bit flag called 'H-flag' contained in each MC defines the shape of an organism. The SR unit does not have the number of MCs included in a single organism; it is not possible for the SR unit alone to determine the number of cycles required to complete a SR process. A counter is necessary to stop the SR process at an appropriate time.

FSM: While a Ubimanager provides a GUI environment for design implementation, it cannot compile from high-level languages such as C or VHDL; the

entire circuit must be implemented by a combination of circuits available in the LUT mode of the Ubicells. The authors resorted to utilize commercially available RTL synthesizer tools to implement the FSM. First, the state chart was converted to HDL using Mentor Graphics HDL Designer. Next, Precision RTL, also by Mentor Graphics was used to syntesize the HDL and produce the RTL schematics with look-up tables (LUTs). The contents of the LUTs as well as the connections among the LUTs were then configured manually to each Ubicell using Ubimanager.

Routing, Floor planning: Figure 6 shows the implemented system. One can see the wiring for routing, and configured Ubicells in this figure. All the routing and floor planning are conducted manually; there is no automatic tool available thus planning of the location of each functional cells, connections among cells, and overall floor-planning is a very crucial part of design implementation on Ubichips, and should be conducted carefully.

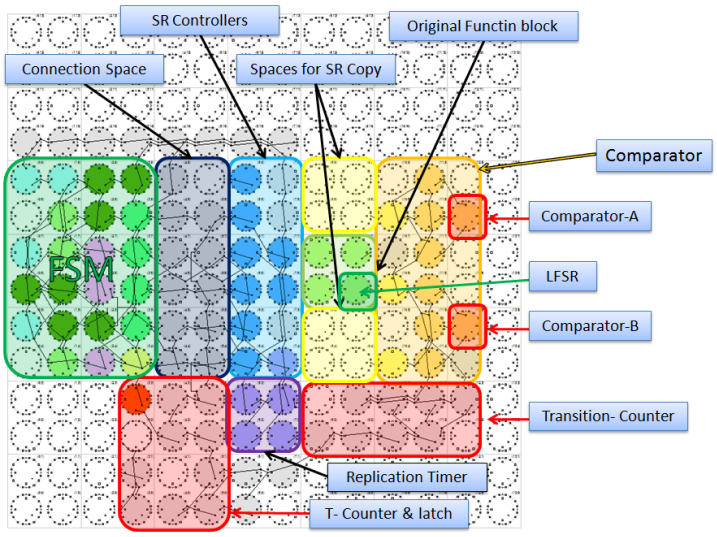

Fig. 6. System Implementation on Ubichip

5 Implementation Results

The implementation was tested using the Modelsim tool integrated in the Ubimanager environment. Figure 7 and 8 show screen-shots of the system under simulation. One can see how a different counter value results in a different number of copies. Each system block was confirmed to be working according to the

design intention. After the system was verified by simulation it was physically implemented in the Ubichip available in the Ubidule board.

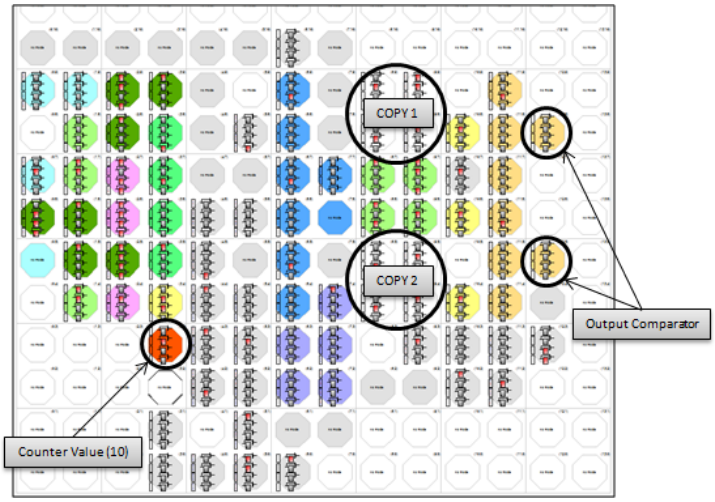

Fig. 7. Simulation View. 2FU mode

5.1 Cell Count, Area Overhead

Table 1 shows the number of cells used for each system block. The fault tolerant system takes a total of 64 Ubicells. The area overhead in this application is significant because a very primitive 4-cell single MC was used as the FU. When a larger FU is implemented, the area of the rest of the system remains unchanged. As the size of Ubicell array in Ubichip is 10 by 10 MCs (20x20=400 Ubicells), the area overhead of this fault tolerant system is 16% of the total array area.

Block:	# of cells:
Control FSM	22
SR Controller (x3)	12
Output Comparator	11
SR Timer	5
Transition Counter	14
Functional Unit	4

Table 1. Cell Count of the Design

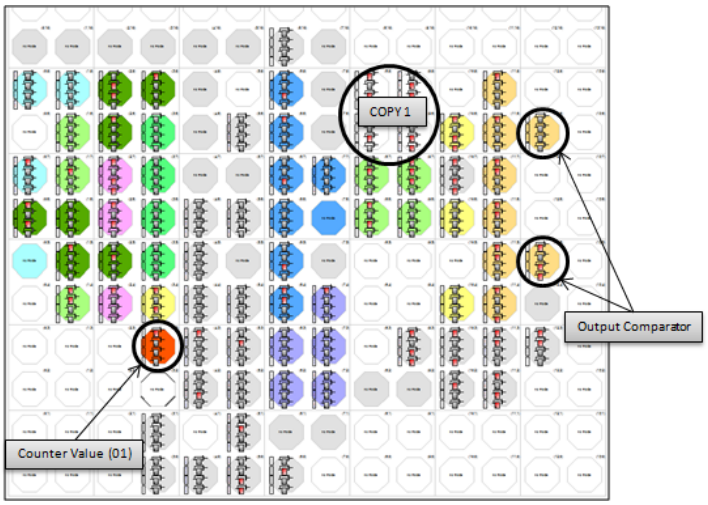

Fig. 8. Simulation View. 1FU mode

5.2 Timing observation

Configuring cells using a serial register means that the time required for configuration increases as the number of MCs to be replicated increases. Table 2 shows the cycles required for the SR unit to complete for different number of MCs. The worst-case estimation for the operating frequency of Ubichip is 50MHz. Since the operation of the FU must pause during the replication process, the replication time especially for larger FUs may become a serious issue for timing-critical applications.

Number of MCs:	Clock Cycles:	Time in seconds:
1	547	10.9μs
2	1,072	21.4μs
10	5,272	105μs
300	157,522	3.15ms

Table 2. Clock Cycle and time in seconds required for Self-Replication (at 50MHz)

6 Conclusion

6.1 Concluding Remarks

Built-in Self-Repair (BISR) is a technique becoming more and more important as the feature size of VLSIs shrink and the chance of faults such as aging defect

and temporal errors increases. In this paper, a conceptual design of a power-aware BISR system using triple-modular redundancy (TMR) was implemented on a custom dynamically reconfigurable platform. Motivation of such system as well as the design and implementation was explained followed by the simulation and implementation results and observation. The authors have successfully demonstrated how the Ubichip, a bio-inspired reconfigurable custom VLSI can be used to implement flexible power-aware fault tolerant systems.

6.2 Future Work

In order to have the fault-tolerant system presented here to be available for more practical uses, the authors have found several directions for future research:

Power estimation: Accurate measurement of power consumption is necessary to have a power-aware system working correctly. As the Ubichip platform does not offer current measurement capabilities, this experiment took transition of output values to estimate the dynamic power consumption of the functional unit. A research should be conducted to incorporate a system to measure the power consumption more accurately.

Error Correction: In this experiment, a simple bit-wise XOR circuit compared the outputs in the TMR system. Further research and development is necessary to implement an error correction capabilities. Such correction system should detect and locate the circuit with error, eliminate the faulty circuit out from the TMR trio, and create a new copy of the circuit in a new location.

SR Controller: The control mechanism of the system was implemented on reconfigurable cells on the Ubichip, resulting in an area overhead on the reconfigurable fabric. A research should be conducted to study the possibility of implementing the self-replication controller circuit as part of the platform so that developers can easily implement this BISR capability in their new designs.

Developing Environment: Ubimanager provides many useful features for designing and implementing functions on Ubichip platform. However, the lack of a high-level language compiler means that the developers must implement LUT contents and routing manually, increasing the development time significantly. Furthermore, the lack of debug tools makes it very time consuming to detect and correct errors in the design. Design tools such as floor planner, interconnect router, high-level language compiler, and debugger would make Ubichip more accesible in practical applications.

7 Acknowledgements

This work has been partially funded by the European Union (PERPLEXUS project, Contract no. 34632).

References

1. M. L. Bushnell and V. D. Agrawal. *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Kluwer Academic Publishers, Boston, 2000.
2. International technology roadmap for semiconductors. 2009 ITRS report, emerging research materials. Technical report, 2010.
3. Richard P. Kleihorst and Nico F. Benschop. Fault tolerant ICs by area-optimized error correcting codes. In *IOLTW*, page 143. IEEE Computer Society, 2001.
4. J.M. Moreno and J. Madrenas. A reconfigurable architecture for emulating large-scale bio-inspired systems. In *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, pages 126 –133, 18-21 2009.
5. André K. Nieuwland and Richard P. Kleihorst. IC cost reduction by applying embedded fault tolerance for soft errors. *J. Electronic Testing*, 20(5):533–542, 2004.
6. PERPLEXUS Project. Pervasive computing framework for modeling complex virtually-unobunded. http://www.perplexus.org/, 2010.
7. Y. Thoma, E. Sanchez, J.M. Moreno, and G. Tempesti. A dynamic routing algorithm for a bio-inspired reconfigurable circuit. In *Field-Programmable Logic and Applications, P.Y.K Cheung, G.A. Constantinides, J. T de Sousa (eds.)*, pages 681–690. Springer-Verlag, 2003.
8. Y. Thoma and A. Upegui. Ubimanager: A software tool for managing ubichips. In *Adaptive Hardware and Systems, 2008. AHS '08. NASA/ESA Conference on*, pages 213 –219, 22-25 2008.
9. Yann Thoma, Andres Upegui, Andres Perez-Uribe, and Eduardo Sanchez. Self-replication mechanism by means of self-reconfiguration. In *ARCS '07 - 20th International Conference on Architecture of Computing Systems 2007*, page Ch. 13. VDE VERLAG GMBH, 2007.
10. Andres Upegui, Yann Thoma, Eduardo Sanchez, Andrés Pérez-Uribe, Juan Manuel Moreno, Jordi Madrenas, and Gilles Sassatelli. The PERPLEXUS bio-inspired hardware platform: A flexible and modular approach. *KES Journal*, 12(3):201–212, 2008.
11. J. Soto Vargas, Juan Manuel Moreno, Jordi Madrenas, and Joan Cabestany. Implementation of a dynamic fault-tolerance scaling technique on a self-adaptive hardware architecture. In Viktor K. Prasanna, Lionel Torres, and René Cumplido, editors, *ReConFig'09: 2009 International Conference on Reconfigurable Computing and FPGAs, Cancun, Quintana Roo, Mexico, 9-11 December 2009, Proceedings*, pages 445–450. IEEE Computer Society, 2009.

Bibliography

[1] Massimo Baleani, Alberto Ferrari, Leonardo Mangeruca, Alberto L. Sangiovanni-Vincentelli, Maurizio Peri, and Saverio Pezzini. Fault-tolerant platforms for automotive safety-critical applications. In Jaime H. Moreno, Praveen K. Murthy, Thomas M. Conte, and Paolo Faraboschi, editors, *CASES*, pages 170–177. ACM, 2003.

[2] R. Baumann. The impact of technology scaling on soft error rate performance and limits to the efficacy of error correction. In *Electron Devices Meeting, 2002. IEDM '02. Digest. International*, pages 329 – 332, 2002.

[3] Keith Bowman, James Tschanz, Chris Wilkerson, Shih-Lien Lu, Tanay Karnik, Vivek De, and Shekhar Borkar. Circuit techniques for dynamic variation tolerance. In *DAC '09: Proceedings of the 46th Annual Design Automation Conference*, pages 4–7, New York, NY, USA, 2009. ACM.

[4] M. L. Bushnell and V. D. Agrawal. *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Kluwer Academic Publishers, Boston, 2000.

[5] Carl Carmichael, Earl Fuller, Phil Blain, and Michael Caffrey. Seu mitigation techniques for virtex fpgas in space applications. Technical report, Los Alamos National laboratory and Novus Technologies, Inc. and Xilinx, Inc.

[6] Maya Gokhale, Paul Graham, Darrel Eric Johnson, Nathan Rollins, and Michael J. Wirthlin. Dynamic reconfiguration for management of radiation-induced faults in fpgas. In *IPDPS*. IEEE Computer Society, 2004.

[7] IEEE. *IEEE standard for space applications module, extended height Format E form factor*. Institute of Electrical and Electronics Engineers, New York, NY, 1995.

[8] International technology roadmap for semiconductors. 2009 ITRS report, emerging research materials. Technical report, ITRS, 2010.

[9] Barry W. Johnson. *Design and Analysis of Fault Tolerant Digital Systems*. Addison-Wesley Publishing Company, Inc., 1989.

[10] Kotaro Kobayashi, Juan Manuel Moreno, and Jordi Madrenas. Implementation of a power-aware dynamic fault tolerant mechanism on the ubichip platform. In Gianluca Tempesti, Andy M. Tyrrell, and Julian F. Miller, editors, *Evolvable Systems: From Biology to Hardware. 9th International Conference, ICES 2010*. International Conference on Evolvable Systems, Springer., September 2010.

[11] P.K. Lala and K.K. Bondali. On biologically-inspired design of fault-tolerant digital systems. In *Electronic Design, Test and Applications, 2002. Proceedings. The First IEEE International Workshop on*, pages 287 –290, 2002.

[12] I. Lemberski and V. Gopeyenko. Method of finite state machine optimal implementation targeting look-up-table architecture. In *Computer Modelling and New Technologies*, volume 11-4, pages 40–46, Riga, Latvia, 2007. Transportation and Telecommunication Institute.

[13] F.P. Mathur. On reliability modeling and analysis of ultrareliable fault-tolerant digital systems. *Computers, IEEE Transactions on*, C-20(11):1376 – 1382, nov. 1971.

[14] J. Manuel Moreno. Specification of the ubicell. Ver 1.2.4. Perplexus Project. Contract Number: 034632, March 2008.

[15] J.M. Moreno and J. Madrenas. A reconfigurable architecture for emulating large-scale bio-inspired systems. In *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, pages 126 –133, 18-21 2009.

[16] VP NELSON. Fault-tolerant computing - fundamental-concepts. *Computer*, 23(7):19–25, July 1990.

[17] André K. Nieuwland and Richard P. Kleihorst. IC cost reduction by applying embedded fault tolerance for soft errors. *J. Electronic Testing*, 20(5):533–542, 2004.

[18] Society of Automotive Engineers. Sae standard. http://www.sae.org/standards/.

[19] International Standard Organization. Iso. http://www.iso.org/iso/home.htm.

[20] M. Patyra and W. Maly. Circuit design for built-in current testing. In *Custom Integrated Circuits Conference, 1991., Proceedings of the IEEE 1991*, pages 13.4/1 –13.4/5, 12-15 1991.

[21] V.B. Prasad. Fault tolerant digital systems. *Potentials, IEEE*, 8(1):17 –21, feb 1989.

[22] PERPLEXUS Project. Pervasive computing framework for modeling complex virtually-unobunded. http://www.perplexus.org/, 2010.

[23] Jan M. Rabaey, Anantha P. Chandrakasan, and Borivoje Nikolia. *Digital Integrated Circuits: A Design Perspective*. Pearson Education, 2003. ISBN: 8178089912.

[24] Ashok K. Sharma. *Programmable Logic Handbook. PLDs, CPLDs, and FPGAs*. Number ISBN 0-07-057852-4. McGraw-Hill, 1998.

[25] P. Shivakumar, M. Kistler, S.W. Keckler, D. Burger, and L. Alvisi. Modeling the effect of technology trends on the soft error rate of combinational logic. In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pages 389 – 398, 2002.

[26] V. Stopjakova and H. Manhaeve. Ccii+ current conveyor based bic monitor for iddq testing of complex cmos circuits. In *European Design and Test Conference, 1997. ED TC 97. Proceedings*, pages 266 –270, 17-20 1997.

[27] Y. Thoma, E. Sanchez, J.M. Moreno, and G. Tempesti. A dynamic routing algorithm for a bio-inspired reconfigurable circuit. In *Field-Programmable Logic and Applications, P.Y.K Cheung, G.A. Constantinides, J. T de Sousa (eds.)*, pages 681–690. Springer-Verlag, 2003.

[28] Y. Thoma and A. Upegui. Ubimanager: A software tool for managing ubichips. In *Adaptive Hardware and Systems, 2008. AHS '08. NASA/ESA Conference on*, pages 213 –219, 22-25 2008.

[29] Yann Thoma, Andres Upegui, Andres Perez-Uribe, and Eduardo Sanchez. Self-replication mechanism by means of self-reconfiguration. In *ARCS '07 - 20th International Conference on Architecture of Computing Systems 2007*, page Ch. 13. VDE VERLAG GMBH, 2007.

[30] Andres Upegui, Yann Thoma, Andres Perez-Uribe, and Eduardo Sanchez. Dynamic Routing on the Ubichip: Toward Synaptogenetic Neural Networks. volume 21, page 228. NASA/ESA Conference on Adaptive Hardware and Systems, 2008.

[31] Andres Upegui, Yann Thoma, Eduardo Sanchez, Andrés Pérez-Uribe, Juan Manuel Moreno, Jordi Madrenas, and Gilles Sassatelli. The PERPLEXUS bio-inspired hardware platform: A flexible and modular approach. *KES Journal*, 12(3):201–212, 2008.

[32] Inc. Xilinx. Xilinx space grade virtex-4qv FPGAs. Webpage http://www.xilinx.com/products/v4qv/index.htm.