University of Cincinnati

Date: 10/28/2015

I, Harsha Vardhan Lakkaraju, hereby submit this original work as part of the requirements for the degree of Master of Science in Electrical Engineering.

It is entitled:
DPA Resistant Logic Arrays for Security Applications

Student's name: Harsha Vardhan Lakkaraju

This work and its defense approved by:

Committee chair: Ranganadha Vemuri, Ph.D.

Committee member: Wen-Ben Jone, Ph.D.

Committee member: Carla Purdy, Ph.D.
DPA Resistant Logic Arrays for Security Applications

A thesis submitted to the

Division of Research and Advanced Studies
University of Cincinnati

In partial fulfillment of the requirements
for the degree of

Master of Science

In the Department of Electrical Engineering and Computing Systems of the College of Engineering and Applied Science
University of Cincinnati
October 2015

By

Harsha Vardhan Lakkaraju

B.E. Electronics and Communication Engineering
Andhra University, India
May 2011

Thesis Advisor and Committee Chair: Dr. Ranga Vemuri
ABSTRACT

Field Programmable Gate Arrays (FPGAs) are becoming an integral part of IC's due to their increased popularity with advancements of technology and due to their various range of applications. Modern FPGAs have less power consumption, complex design size and more predictable project cycle compared to ASICs. Advancements in this technology has brought exponential growth in the use of embedded systems such as smart cards, laptops, tablets, mobile phones and other mobile technologies. This has resulted in the need for sophisticated and secured hardware implementation of cryptographic algorithms. Even though these algorithms are secured mathematically they are not effective when implemented in hardware due to side channel leakage. This provides an attacker a medium to attack the device, commonly known as side channel attacks (SCA).

Differential Power Attack (DPA) is widely used medium of hardware attack. DPA attacks measure power levels at different instances of the chip and apply statistical analysis to overcome many countermeasures, such as added noise that obscures low level bit values. Measuring the power usage can identify the computational operations of the device. An analysis will reveal several bits of the crypto-key at a time; the process is repeated to eventually produce the entire key. DPA attacks are dangerous and powerful because they circumvent the hardware and software security that vendors have put in place. Because such attacks can be passive and non-invasive, it is possible for an intruder to compromise an embedded system without leaving a trace.

Several countermeasures have been proposed to prevent DPA attacks at both High and Low Levels of abstraction. One such countermeasure is Secure Differential Multiplexer based Logic (SDMLp). Cellular Array based FPGA architecture has been implemented to realize the secured Logic Array in this work.

In this thesis, we propose a new flow for a DPA resistant Logic Arrays using Reduced Ordered Binary Decision Diagram (ROBDD) approach using SDMLp logic style. Using the proposed design flow, an FPGA architecture is realized with one SDMLp cell (implemented as MUX) functioning as PLB. Two different architectures are proposed; the Fixed Array architecture that makes use of via and the Programmable Logic Array Architecture that is programmed using serial bit-stream inputs. In both cases, the coefficient cofactor vs. the key guesses and total number of vectors are analyzed, and the results show that the design is DPA resistant.
To My Dearest Parents
ACKNOWLEDGMENTS

My humble thanks to the almighty for his blessings throughout my life.

My first and foremost thanks to Dr. Ranga Vemuri for his guidance both as an advisor and a professor. My journey in the DDEL lab would be incomplete without his support. I've always has utmost respect and awe for his enormous knowledge in any topic and enjoyed the discussions we both had on and off university.

I would like to thank Dr. Jone and Dr. Purdy for graciously accepting my thesis committee request. I have learnt a lot in different topics in VLSI and the class sessions have been enjoyable.

My DDEL colleagues, especially seniors have been kind enough to give me insights on various topics and have been helpful and informative throughout my research work. I would like to thank Mike, Manoj, Lakshmi and Antar for providing the background work for this thesis. I would also like to thank Arun and Karthik for their inputs and help. I would also like to thank Mike and Lijitha for taking time to proof read my thesis. DDEL life wouldn't have been fun and enjoyable if not for my colleagues and juniors.

I would not have been here if not for the support and love of my parents Sudha Rani and Madan Mohan. I am grateful for all the sacrifices they made to make my life comfortable and I owe everything to them. Life wouldn't be the same without my brother Siddhu. I am thankful for all your love and showing me the path to become what I am today. I wish you all the best for your future. My sincere thanks to all my family members. I promise I will strive hard and work harder to be a better person and make you feel proud and happy.

I would like to thank all my friends both in the US and in India for their love and support. Special thanks to Nivedita for constant support and encouragement. Thank you for being there for me and for watching over me as I progressed over years from a school kid to Masters Graduate.
# Table of Contents

1. Introduction .......................................................................................................................... 1  
1.1 FPGA - Overview .............................................................................................................. 1  
1.2 Embedded Devices Security ............................................................................................. 1  
1.3 Various Cryptography Attacks  
  1.3.1 Side-Channel Attacks ................................................................................................ 4  
  1.3.2 Countermeasures against DPA .................................................................................. 7  
1.4 Differential and Dynamic Logic ....................................................................................... 8  
1.5 Existing DDL Methodologies  
  1.5.1 Sense-Amplifier Based Logic .................................................................................... 10  
  1.5.2 Simple-Dynamic and Differential Logic .................................................................... 10  
  1.5.3 Wave-Dynamic and Differential Logic ....................................................................... 11  
  1.5.4 Secure Differential Multiplexer Logic using Pass Transistors ............................... 12  
1.6 Thesis Overview ................................................................................................................ 14  

2. Cellular Array based FPGA ............................................................................................... 16  
2.1 FPGA architectures  
  2.1.1 FPGA Architectures Based on PLB .......................................................................... 16  
  2.1.2 Programming Technologies ..................................................................................... 18  
  2.1.3 FPGA Routing Architectures ................................................................................... 19  
2.2 CA based FPGA Architecture .......................................................................................... 20  
2.3 PLB using SDMLp for proposed Architecture .................................................................. 23  
2.4 Routing Structure for proposed Architecture ................................................................... 24  
2.5 Summary ............................................................................................................................ 27  

3. Technology Mapping and Proposed Synthesis Flow ....................................................... 28  
3.1 Binary Decision Diagram – Overview ............................................................................. 28  
3.2 Reduced Ordered BDD overview ..................................................................................... 29  
3.3 Variable Order – Impact .................................................................................................. 31  
3.4 SDMLp to BDD – Mapping ............................................................................................. 33  
3.5 Proposed Flow .................................................................................................................. 36  
3.6 BDD Synthesizer Module ............................................................................................... 37  
3.7 Modified BDD Structure (MBDD) Module ..................................................................... 39  
3.8 Technology Mapping Module ........................................................................................ 40  
3.9 Summary ............................................................................................................................ 43  

4. Experimental Results and Analysis ................................................................................... 44  
4.1 Experimental Setup ......................................................................................................... 44  
4.2 Design Metrics Evaluation for Proposed Flow ............................................................. 44  
4.3 Design Metrics for Programmed FPGA Architecture .................................................. 50  
4.4 Security Metrics Evaluation for Proposed Flow ............................................................. 56  
4.5 DPA attack on DES ......................................................................................................... 68  
4.6 Conclusions and Analysis ............................................................................................... 71
Chapter 1

Introduction

1.1 FPGA Overview

An integrated circuit is a semiconductor material made from Silicon. It consists of a set of electronic components in a package referred to as a chip or an IC. IC’s are integral part of everyday life. They are used in electronic products like laptops, phones, high performance computing etc. They are categorized as (based on cost, tools availability, performance and design flexibility) ASIC’s - Application Specific Integrated Circuits and FPGA’s - Field Programmable Gate Arrays.

While ASIC’s are customized for a specific use rather than a general-purpose application, FPGA’s are designed to be configured by a customer or a designer after manufacturing. FPGAs consist of configurable logic block network, and reconfigurable interconnects that allow the blocks to interact with each other, and can be configured to perform any complex function, or merely simple logic gates implementation. Logic blocks are made up of simple memory cells like Flip-Flops or more complex blocks of memory.[1]

FPGA’s have faster time to market, less recurring expenses, simpler design cycles, easier to predict project cycles, field re-programmability and are more reusable compared to their ASIC counterparts making them more popular.

Due to their increased popularity and various ranges of applications FPGA’s are becoming an integral part of large mixed systems. They are used in variety of applications including embedded applications, where security is a concern (e.g. smart cards, secure communication protocols for data transfer over non-secure medium etc.). These devices have seen increased security threats off late and recent research has emphasized on overcoming these attacks.

1.2 Security of Embedded Devices

Embedded systems provide critical functions that can be sabotaged by malicious entities. There are many entities involved in a typical embedded system design, manufacturing, and usage chain. Security requirements vary depending on whose perspective we consider.
Implementing cryptographic algorithm in software and hardware layer improves but not necessarily ensures security in embedded systems. The algorithms encrypt the data for transferring over an unsecure medium. These cryptographic algorithms use secret key to convert original message to encrypted message (cipher text – encryption). The original text is obtained from cipher text using the same key and this process is decryption process. Broadly, cryptographic algorithms can be classified into three classes—symmetric ciphers, asymmetric ciphers, and hashing algorithms, which are briefly described below [3]:

- **Symmetric ciphers** require the sender and receiver to use the same secret key to encrypt and decrypt data. Symmetric block ciphers include DES, 3DES, and AES blocks.

- **Asymmetric ciphers** (also called public-key algorithms), on the other hand, use a private (secret) key for decryption, and a related public (non secret) key for encryption or verification. They are typically used in security protocols for verifying certificates that identify communicating entities, generating and verifying digital signatures, and for exchanging symmetric cipher keys. Examples of Asymmetric ciphers include RSA, Diffie-Hellman etc.

- **Hashing algorithms** such as MD5 and SHA provide ways of mapping messages (with or without a key) into a fixed-length value, thereby providing "signatures" for messages. Thus, integrity checks can be performed on communicated messages by (a) having the sender “securely sending” the actual hash value of a message along with the message itself, (b) allowing the receiver to compute the hash value of the received message, and (c) comparing the two signatures to verify message integrity.
1.3 Types of Cryptography Attacks

The Cryptography attacks are classified into three main categories [4]

- **Software Attacks:** They are carried out through software agents like Trojans, viruses or worms etc.

- **Invasive (Physical) Attacks:** Physical intrusion into the system either at system, board or chip level is required along with skilled attacker and expensive equipment.

- **Non-Invasive (Side Channel) Attacks:** These attacks analyze system variables like power consumption, faults and extracts the key using correlation.

The following figure (Fig. 2) shows the types attacks performed on Embedded Systems.

![Embedded System Attacks Diagram](image)

Figure 2: Embedded Systems Attacks. Source [4]
1.3.1 Side Channel Attacks

Side-channel attack is a type of physical attack in which an external entity tries to exploit physical information leakages such as power, timing, or electromagnetic radiation. They are non-invasive and passive and performed using relatively cheap equipment. Due to these limitations, they pose a serious threat to the security of most cryptographic hardware circuits. Such devices range from personal computers to small-embedded devices such as smart cards and RFIDs (radio frequency identification devices).

In this work, we are interested in power-based Side Channel Attack analysis and the focus is made on Simple Power Analysis (SPA) and Differential Power Analysis (DPA).

Power-Attacks Analysis

CMOS circuits Power Consumption: There are three distinct dissipation sources in Static CMOS. The first one is due to the leakage currents in transistors, the second one is due to the short-circuit currents which exits for short period due to simultaneous conduction of CMOS and NMOS transistors during switching. Finally, the Dynamic power consumption is due to the charge and discharge of the load capacitance $C$. Technology scaling affects the effective scaling of importance of the mentioned dissipation sources. But the dynamic power consumption is important from a side-channel perspective since it determines the relationship between a device’s internal data and its externally observable power consumption [3].

![CMOS Inverter Dynamic Power](image)

$\text{Energy/transition} = C_L \cdot V_{dd}^2$

$\text{Power} = \text{energy/transition} \cdot f = C_L \cdot V_{dd}^2 \cdot f$

Figure 3: CMOS Inverter – Dynamic Power. Source [3]
Thus Dynamic power can be expressed as:

\[ P_{\text{dyn}} = C_L V^2_{\text{DD}} P_{0\rightarrow1} f \]

where, \( P \) is the switching activity, \( P_{0\rightarrow1} \) is the probability of a \( 0 \rightarrow 1 \) transition, the working frequency of the system is \( f \), and \( V_{\text{DD}} \) is supplied system voltage. In CMOS devices, the highest peak will appear during the charge of the capacitance (i.e., \( 0 \rightarrow 1 \) event) while measuring the power consumption, which is done either at the ground pin or at the power pin. During the discharge, short-circuit current can be measured accurately. This data-dependent power consumption is the origin of side-channel information leakages.

**Simple Power Analysis (SPA)**

Simple power analysis (SPA) deduces the operational information of the device by interpreting the power consumption of the device. This is explained using the example in Fig. 4, which shows the power consumption trace of a device performing an AES (advanced encryption standard) encryption [6]. It can be seen clearly that the pattern is repeated 10 times, which corresponds to the 10 rounds of the AES when implemented in its 128-bit version.

![SPA traces from an AES encryption. Source [5]](image)

AES 128-bit version has 10 rounds, and knowing that a device is performing an AES encryption does not expose its secrets at all. However, such a visual inspection of the leakage traces may be the preliminary step in a more powerful attack, i.e. by determining the parts of the traces that are relevant to the adversary. Modular exponentiation performed with a square and multiply algorithm is a good example in which the sequence flow is dependent on the data. If the square operation is implemented differently than the multiple operation – a tempting choice, as this will allow specific optimizations for the square operation, resulting in faster code – and provided this difference results in different consumption patterns, then the power trace of an exponentiation directly yields the (secret) exponent’s value. Broadly
speaking, all instructions consisting of conditional branch operations that depend on secret parameters are at risk of being attacked.

**Differential Power Analysis (DPA)**

DPA is fast and accurate and is proven to be one of the best side channel attack technique implemented to extract information from a device [5]. Unlike SPA, DPA performs mathematical correlation on the current traces that are obtained from the various operations performed on the circuit [7][8][9]. The following section explains the DPA attack on a DES platform.

The cipher text, key and the encrypted text are done based on 16 rounds of encryption.

Function \( f \) is characterized by the 8 S-Boxes. Each S-Box is a combinational look-up table with 6-input 4-output logic. During the \( i^{th} \) iteration, the 48-bit sub key is XORd with \( R_{i-1} \). The output is then divided into eight blocks of six inputs each and fed into S-boxes. Cipher text is obtained from the final round \( R_{16}L_{16} \).

In a DPA attack, analyzing the S-Boxes for all the blocks returns the 6-Bit key. DPA attack on DES consists of obtaining current traces for plaintexts to cipher-text conversion operations (N) with each of N operations containing K sample traces.

If the traces are uncorrelated, the differential traces are flattened out for incorrect guesses. If the key guess was correct, the trace would have been divided based on value of \( b \), and thus, it will have a peak indicating the correlation to the key. This is repeated in all the S-Boxes till the key is obtained. The differential trace of DPA attack on DES is shown in the following figure (Fig. 5).

![Figure 5: DES based DPA attack showing current traces. Source [10]](image-url)
1.3.2 Countermeasures against DPA attacks

Devices that are attacked using DPA based on the two weaknesses of hardware implementations mentioned below:

- Secret Key is related to the output of the algorithm, and this results in easy trace back (algorithmic weakness)

Thus, existing countermeasures aim at thwarting DPA attacks at these levels (algorithmic or circuit).

Algorithmic Level Countermeasures

Conditional Jumps/Branches in the algorithms have dependencies between secret key and data. These countermeasures work on these dependencies. It is implemented by adding random set of instructions causing the results to show random power consumptions. Here, measures should be taken to make sure that these distributions are randomized so that correlation cannot weed out the traces.

Another technique is based on masking all or part of the input vectors and convert them into an intermediate value and encrypted. High Order DPA (HODPA) is proven to be efficient against masking [11].

FSM modification to break common side channel attack models is another technique implemented as an algorithmic level countermeasure [32][33][34][35].

Circuit Level Countermeasures

Here, the current traces are made non-correlated to input data/key at both cell and architectural level by modifying the circuit. In an architectural level technique, operations are masked causing random power consumptions using LFSR techniques [12]. DVFS [13] technique changes voltages randomly to make the circuit DPA resistant.
Cell level countermeasures

Here, work on eliminating the variations dependent on power. It aims at making the power consumptions of each cell (gate) to be the same irrespective of applied inputs. Constant current consumption is achieved using DDL logic style that are implemented using SCMOS.

Several countermeasures have been implemented in circuit level to remove this dependency which aim at designing dedicated logic styles to prevent DPA by making current dependency independent of input data. They are reviewed in the following section.

1.4 Dynamic and Differential Logic

A necessary condition for reducing data dependency is obtained by ensuring equal capacitance charged/discharged for each clock cycle, which is obtained by implementing Dynamic and Differential Logic (DDL). Constant current is consumed for every cycle if the following conditions are satisfied,

- Signals are represented in original and complement forms making the logic differential.
- The circuit operation is divided as evaluation and pre-charge phases in dynamic logic implementation.

During pre-charge, both signals are discharged to 0, and in evaluation phase one of the signal transitions to high due to differential property. Thus, at least one 0 to 1 transition is ensured for every clock cycle, which can occur either at original or complementary signal [14][15].

Other important constraint is to maintain constant current consumption by ensuring same amount of capacitance is charged or discharged during every clock cycle. Thus, the effective capacitance (consisting of intrinsic, output load and interconnect capacitance) of the original and complementary portions of the circuit are matched.

If the inputs make the transition from state 00 to state 01 and finally to state 11, the outputs in each case make transitions from state 0 to 1 to 1 and finally to 0.
Figure 6: XOR gate using SCMOS Logic. Source [16]

Figure 7 shows a black-box representation of XOR gate using arbitrary DDL logic style. In the evaluate phase, either of the two output signals change from 0 to 1 and the outputs transition to 0 during pre-charge phase. This ensures that for every clock cycle there is a 0 to 1 transition and 1 to 0 transition in either original or complementary portion.

Figure 7: XOR Gate implemented using DDL Logic. Source [16]
1.5 Existing DDL Methodologies

There are 3 main DDL based logics and are mentioned in the following section;

1.5.1 Sense Amplifier Based Logic (SABL)

The differential and dynamic signaling is ensured using dynamic CMOS logic, which consists of differential pull-down network and is implemented so that all internal nodes are connected to any one of the output nodes. Using static inverters between two stages results in SABL having cascading problems (like dynamic CMOS logic). They are highly sensitive to glitches and are constrained to only non-inverting logic usage.

1.5.2 Simple Dynamic and Differential Logic (SDDL)

Static CMOS gates are used to implement the dynamic and differential logic. It is implemented using DeMorgan's theorem, the equations are shown below,

\[
\begin{align*}
(A + B)' &= A'.B' \\
(A.B)' &= A' + B'
\end{align*}
\]

SDDL uses clock signal to push the circuit into pre-charge state. Figure 8 shows an AND gate using SDDL logic. Pre-charge signal is 0 during pre-charge state and 1 during evaluation state. Only AND and OR gates are used to implement this logic, and this results in higher area and power after synthesis when compared to SCMOS. The global pre-charge also has very high load capacitance since it has to be routed to all gates, it has high load capacitance resulting in timing mismatches due to skews in race conditions.

![Figure 8: SDDL Implementation of AND gate. Source [15]](image)

1.5.3 Wave Dynamic and Differential Logic (WDDL)

In this logic, Static CMOS logic is used to realize dynamic and differential logic. Only AND-OR Logic gates are used in WDDL logic. Where the SCMOS OR gates are
replaced by SCMOS AND gates and complementary inputs are fed and vice-versa.

It uses a pre-charging technique to propagate a wave of 0's through the inputs of all static gates, and since it uses only positive gates, the pre-charge wave forces the gates to a state of 0 (pre-charge state). During this phase, WDDL AND inputs are forced to 0 using pre-charge circuit or a propagated 0-wave. During evaluate phase, the inputs are made complementary to push the gates to evaluation phase.

![Figure 9: WDDL AND Gate. Source [15]](image)

Figure 9 shows the implementation of an AND gate in WDDL logic. Since this logic uses only positive gates it poses a constraint during synthesis process, thereby, increasing the area and power consumption compared to SCMOS.

**1.5.4 Secure Differential Multiplexer Based Logic using Pass Transistors (SDMLp)**

The SDMLp [14] logic style was developed in Digital Design Environments Lab (DDEL) in part by Lakshmi Narasimhan Ramakrishnan, as a part of an ongoing research on Side Channel Attack Resistant designs for enhanced IC security, and it is based on Complementary Pass Transistor Logic (CPL) [18]. CPL is a differential logic and has symmetrical structure, which is the base for a secure design. The only constraint CPL has is that it cannot guarantee one switching per cycle, as it is only differential but not dynamic. Figure 10 shows a transistor level implementation of CPL.
SDMLp logic is based on CPL and it works on the two basic principles states: evaluation and pre-discharge. NMOS transistors make up for the Evaluation state.

A generic 2-input SDMLp cell is implemented and is depicted in the following figure (Fig 11), which can be configured based on Table 1.1 to realize any two input function. The inputs are complementary and fed to the pass transistors.
The above functions can be realized using an SDMLp cell by changing the corresponding inputs and select signals.

NMOS network consists of transistors m(1,2,7,8) and is used for evaluation and the transistors m(3,4,9,10) are used to propagate pre-discharge signal during setup.

Pre-Discharge Phase:

In this phase the select signals are forced to logic low (0). The NMOS network is turned off and thus they stop conducting, while the PMOS network is turned on and the transistors start conducting. The output signals are forced to logic 0.

Evaluation Phase:

In this phase, signals S and Sbar return to complementary states. In this phase valid out and outbar signals are obtained as there is no direct path through the pre-discharge network.

SDMLp has near constant current consumption which improves its attack resistance and it also consumes relatively less power and occupies less area when compared to the WDDL logic style. However, it incurs a delay penalty when compared to SCMOS style [37].
1.6 Thesis Proposal and Overview

The importance of FPGAs and the need for Hardware Security has been established in the previous sections provides the motivation for this thesis work. SDMLp logic shows significant improvement in current variance (a common security metric) when compared to existing logics.

It has a limitation that it can be configured only for 2 input functions, causing scaling problem when integrated to FPGA design flow resulting in higher number of cells used. In this work, we propose a synthesis flow for SDMLp using Binary Decision Diagram (BDD) to implement FPGA architecture.

In this thesis, we have adapted a secure design flow for a Cellular Array based FPGA architecture using SDMLp logic. The PLB design for the proposed FPGA architecture has been introduced, which is a SDMLp based MUX. The routing structure for the proposed architecture is introduced with DFF and two Transmission gates connected as a scan chain and programmed to implement any function. The synthesis flow for FPGA is proposed and evaluated on the basis of design metrics and security metrics, with an aim to make this flow hardware secured FPGA flow.

In Chapter 2, we discuss in detail the backbone of the FPGA architecture. We introduce the Cellular Array based FPGA design. Detailed structure of the CA based FPGA architecture (PLB and Routing) is explained.

In Chapter 3, Technology Mapping for the proposed FPGA is explained using Binary Decision Diagrams. There we discuss the impact of Variable Order on the size of the BDD, and also discuss the reduction of BDD to obtain a canonical representation of Boolean Function.

In Chapter 4, the proposed synthesis flow is described in detail. Tree reconstruction Approach for mapping is explained and its importance in obtaining CA based FPGA architecture is explained. The major modules involved in synthesis flow are discussed, and also we show how to obtain an optimal variable order using existing BDD packages. The Algorithms used for developing BDD reduction are analyzed and technology mapping of the obtained BDD tree structure to FPGA architecture is discussed.

In Chapter 5, we evaluate performance of the SDMLp design (in terms of current variance and standard deviation) compared to WDDL for few functions. Next, we design DES cryptographic circuit and evaluate security (maximum instantaneous current variance) of the design. We then attempt a DPA attack on DES circuit and the experimental results are presented in this chapter.
Chapter 6 discusses further improvements that can be done in this work by introducing sequential circuitry in the flow and finally concludes with pointers to future work.
Chapter 2

Cellular Array Based FPGA

2.1 FPGA Architectures

Field programmable Gate Arrays [25] are a type of IC’s that are pre-fabricated and can be made to realize any circuit or digital system by programming any number of times. Normally FPGA’s comprise of the Programmable Logic Block (PLB) where the functionality is implemented, the Routing Network that connects the PLBs and the IO pins that account for off-chip connections.

FPGA’s can be classified into various types based on PLB’s, programming technologies or routing architectures.

2.1.1 FPGA Architectures Based on PLB’s

A programmable logic block (PLB) is a basic component where the logic and storage functionality for any design is carried out. This is realized if the basic component is either a transistor or an entire processor. In a fine-grained architecture (where the logic blocks are transistors) trade off is made since large amount of programmable interconnect is required. Thus having an undesirable overhead on area and power consumption resulting in degraded performance. In the tree-based architecture that consists of processors as logic blocks, the design is very complex (course-grained) and cannot be used to implement simpler functions, which results in overhead in area and power resources.

They are various PLBs used in FPGA architectures [25],

• Look-up Table (LUT): This architecture maintains a trade-off between fine-grain and coarse-grained systems. A single Basic Logic Element is used as a PLB, or clusters of locally interconnected BLEs are realized to form a PLB. BLE consists of a LUT, and a memory element. A 4 input Look-Up table based PLB is implemented and is shown in Figure 12. The LUTs can be used to implement any n-input function and it can be configured using 2n bits.
• MUX based: The PLB of a MUX based FPGA architecture comprises of logic of interconnected MUX's that perform operations based on the number of inputs. ACTEL ACT1 [26] module is a prime example that makes use of MUX based FPGA architecture. Figure 13 below, shows a Logic Block of a MUX based architecture of ACTEL ACT1 module.
2.1.2 Programming Techniques

In recent years, numerous programming techniques have been used for logic array based architectures. Each technology exhibits different characteristics, which result in significant changes that affect the Logic Array architectures. The following section gives brief description of well-known programming technologies [25].

Programming Technology based on SRAMs: Static RAM memory cells are used here. These FPGAs can be used;

- To program the routing interconnect of FPGAs controlled by smaller Multiplexer networks.
- To program Configurable Logic Blocks (PLBs) to realize any given function.

Programming Technique based on Flash Memory: This programming technology makes use of Flash or EEPROM based cells [25]. This is non-volatile in nature and provides more area and power efficiency over SRAM based. However, they cannot be reprogrammed for infinite number of times.

Programming Technique based on Anti-Fuse: This technology is an alternative to SRAM and flash-based technologies [25]. This technology has lower “on” resistance and parasitic capacitance making it more popular and also utilizes low area. In this technology, the Anti-Fuse cannot be reprogrammed, as they don’t use standard CMOS logic based cells/gates.
2.1.3 FPGA Routing Architectures

The programmable logic blocks provide the computing functionality, but the interactions between these blocks are realized through programmable routing network. Any simple or complex function can be implemented by parsing programmable bits stream into the system using the programmable routing network. Different programmable technologies are used to program the routing network. The two main types of Routing architectures are:

Island Style Routing Architecture: A traditional island-style FPGA architecture (also termed as mesh-based FPGA architecture) is shown in figure 4 [25]. The programmable logic blocks (PLBs) are localized on a two-dimensional grid network and are interconnected through a programmable routing network. The routing network comprises of pre-fabricated wiring segments and programmable switches that are organized in horizontal and vertical routing channels. Routing channel takes about 80% of total space.

Figure 14: FPGA Architecture - MESH based. Source [25]
Hierarchical Routing Architecture: The hierarchy in placement and routing of connections between different logic blocks is achieved using the locality of connections principle followed by most logics [25]. The routing architectures in this logic are divided into separate groups/clusters based on Logic blocks. These clusters are recursively connected to form a tree-based structure.

Figure 15 shows an example of Hierarchical Routing architecture.

![Figure 15: FPGA Architecture TREE based. Source [25]](image)

### 2.2 CA Based FPGA Architecture

Cellular-Array (CA-type) FPGA [27], was especially developed for data-path intensive designs. This fine-grain architecture is composed of a 2-D array of locally connected logic cells and a network of local and global busses for distance connections. The local connectivity is very well suited for the data-path portion of the design but it causes problems for the layout synthesis of the random logic. The routing domain of these FPGAs is much more restricted than, for example routing domain of Look-up Tables FPGAs. Therefore, the separation between logic synthesis and layout synthesis is no longer effective.

Cellular Array architecture based FPGAs are characterized by symmetrical array of locally connected logic blocks. Logic blocks are usually of the fine-grain type and the number of inputs and outputs are limited. The traditional methods of separate
technology mapping, placement and routing used for other FPGAs become of little value for realizing circuits on CA-type FPGAs.

A large number of logic cells is used for wiring connections or left unused in the traditional approach. Therefore, specific logic and layout synthesis approaches are needed to efficiently map circuits onto CA-type FPGAs.

In these FPGAs, the design has to be implemented with predominantly localized connections to achieve better utilization of logic-blocks. For long-range communication local and express buses should be used to reduce the delay.

Figure 16 shows a generic architecture of a CA type FPGA.

Figure 16: CA type FPGA Architecture. Source [27]

Figure 17 shows the black box of the basic PLB used in the FPGA architecture with its inputs and outputs.
Figure 17: PLB Black Box of the proposed FPGA.

Figure 18 shows the complete architecture of the proposed FPGA design.

Figure 18: Proposed CA based FPGA architecture.

The PLBs are connected to each other with a set of local wires and the inputs to the PLB’s are connected using global horizontal wires. There are two sets of vertical wires, one set is VDD and GND lines and the other set is output lines. The proposed FPGA architecture has 100X100 PLB’s and each PLB has local and global routing to
implement any type of logic. The routing wires are connected using DFF cells that can be programmed to implement any function. The proposed FPGA architecture can perform any combinational logic for multi-input multi-output functions and is designed for a maximum of 8 inputs, 8 outputs functions.

The following sections of this chapter describe in detail the CA based FPGA architecture using SDMLp MUX as a PLB and the routing and placement structure for the proposed CA-based FPGA.

2.3 PLB using SDMLp for proposed Architecture

As explained in the previous section, a BDD can be represented as a 2:1 MUX, and as discussed in the first chapter, an SDMLp cell represents a MUX, which can be programmed to perform any logic function with two inputs. As seen from figure 27, a single PLB has 8 input lines connected to the 0 input (A, Abar), 1 input (B, Bbar), S input (S, Sbar), and VDD and GND. It also has 2 output lines (O, Obar). The 0 and 1 inputs are connected to other PLB's through local routing, and the VDD, GND, S, Sbar and output pins are connected through the global routing lines.

The PLB for the proposed architecture is designed to perform two basic functions, MUX and Buffer. The internal connections and PLB cross section is shown in figure 19.

![Figure 19: Cross section of PLB for proposed architecture.](image)

The Inp_0 of the MUX's are connected locally to the PLB vertically below the initial PLB or can be connected to any of the global routing wires (functional inputs or Vdd and GND). The Inp_1 of the MUX's are connected locally to the vertically right adjacent PLB or can be connected to any of the global routing wires. The S and Sbar of the PLB are connected to functional inputs or Vdd and GND lines. The outputs can
feed the top adjacent or left adjacent PLB’s through local routing, or can feed the global routing Out/Outbar wires.

Figure 20: Magic Layout of a single PLB.

Figure 20 shows the Magic layout of a single PLB using SDMLp design.

2.4 Routing Structure for the Proposed Architecture

The CA based FPGA architecture is modeled based on Tree-based (Hierarchical) FPGA architecture. This routing structure takes advantage of the locality by placing locally connected PLBs close to each other. In the proposed architecture, the PLBs are connected either vertically or horizontally to the adjacent PLBs. This is achieved by using ROBDD during synthesis process.

As shown in figure 18, the routing of the proposed architecture is divided into two parts,

• Local Routing: The interconnection between two PLBs is achieved by local routing, which connects the 0-input to the next PLB that is vertically top of
the current PLB or the 1-input to the next PLB that is horizontally left of the current PLB. Apart from the PLB-PLB interactions, local routing is also used to connect the S, Sbar signals of the PLB to the global signal wire inputs, and also to Vdd and GND. Fat Wire Routing has been implemented in the design to make the wire lengths of complementary inputs same to have almost equal capacitances (for security).

- Global Routing: The global routing consists of long wires that connect all the PLB’s. They are divided further as vertical and horizontal global routing signals. There are 4 global vertical wires for each PLB. They are divided into two sets of two wires each. The first set consists of Vdd and GND lines, which are fed to each PLB in that column. The second set consists of Out/Outbar lines that are connected to each PLB. There are 16 horizontal wires that connect entire row of 100 PLB’s. These 16 lines are the source of 8 input signals and their complements.

Having described the PLB’s and the routing architecture of the proposed FPGA, the connections (via) between the routing structures is implemented using scan chain structure consisting of one DFF and two Transmission Gates. The following figure (figure 21) shows vias for one PLB and it’s routing structure.

![Figure 21: Via connections for 1 PLB for proposed FPGA architecture](image)

The dots represent via that can be achieved by using DFF and two transmission gates connected together as scan chain. The via exists if the value of DFF is 1 (the transmission gate is on and thus the via is connected) and not connected if the value of DFF is 0. The following figure 22 shows the routing structure for one input (complementary inputs). The total numbers of such scan chain networks needed for one PLB is 26. Thus, for a 100x100 FPGA network, total number of scan chains required is 260000. The flow generates the Programmable bits that are fed to the scan chain to realize any function.
Figure 22: Routing Structure for one input connection.

The MAGIC layout for the routing structure for one input connection is shown in the following figure 23.
2.5 Summary

In this chapter, the CA based FPGA architecture was discussed. The PLB and the routing structure for the proposed architecture was discussed. In the previous chapters, the motivation for having secure hardware FPGA architecture was discussed. The Fixed Array architecture proposed in this work is synthesized using vias, and the Logic Array architecture proposed in this work is synthesized using scan chains and by streaming programmable bits into the logic array.

In the next chapter, we discuss the technology mapping technique employed using ROBDD synthesis, and tree re-structuring technique employed to realize the architecture. We discuss the proposed design flow for both Fixed Array architecture and Logic Array architecture.
Chapter 3

Technology Mapping and Proposed Synthesis Flow

The SDMLp based Cellular Array FPGA architecture was explained in Chapter 2. The PLB of the proposed architecture is a 2:1 MUX. The motivation for performing ROBDD synthesis and mapping of SDMLp to a BDD and the synthesis flow proposed for Fixed Array and Logic Array architectures is discussed in this chapter.

3.1 Binary Decision Diagram – Overview

Binary Decision Diagram (BDD) Data Structure giving graphical representation of a Boolean Function, which in turn can be represented as a rooted, directed, acyclic graph containing several decision and terminal nodes that interact with each other. [19]

A BDD has the following structure:
A terminal node with out-degree is 0.
A non-terminal node with out-degree of 2 with low and high edges.

![Figure 24: Terminal Nodes and Non Terminal Nodes.](image)

From Figure 24, it can be seen that the BDD has two non-terminal nodes $u$ and $v$. Here, low$(v) = 0$ and high$(v) = 1$ and low$(u) = v$ and high$(u) = 1$. The variable nodes
$u$ and $v$ are represented as $\text{var}(u)$ and $\text{var}(v)$ respectively. Low edge is represented by dashed line and the corresponding high edge is represented by bold line.

### 3.2 Reduced Ordered BDD – Overview

A BDD is said to be an ordered BDD if all paths from the route are ordered and have different variables, i.e. the variables follow a given variable order $x_1 < x_2 < x_3 \ldots < x_n$ for all the paths through the graphs.

![Diagram of Ordered BDD](image)

Figure 25: Ordered BDD

Figure 25 illustrates an ordered BDD with variable order $x_1 < x_2$, and $x_1 < x_3$.

A BDD is reduced if the conditions have been applied to its graph:

- Merge any isomorphic sub-graphs.
- Nodes whose two children are isomorphic are eliminated.

Thus, any BDD is a Reduced Ordered BDD (ROBDD) [20] if it satisfies the following conditions:

- If there are no nodes that are distinct with same input variables and successors.
- There are no Isomorphic nodes with identical high and low successor.
It is depicted in the following figure;

![Figure 26: Isomorphic Nodes and Isomorphic Children.](image)

ROBDD is canonical for a given Boolean expression and a variable order [21] and is used extensively for equivalence checking, technology mapping and logic synthesis.

The Binary Decision tree when transformed to Optimal Reduced Binary tree using the reduction techniques mentioned above is shown in the following figure:

![Figure 27: ROBDD with Variable Order x1<x2<x3. Source [20]](image)
3.3 Impact of Variable Order

The size of the ROBDD is critically dependent on the Variable Order. It can be illustrated with the help of the following example,

Let us consider the boolean function,

\[ f(x_1, \ldots, x_{2n}) = x_1x_2 + x_3x_4 + \ldots + x_{2n-1}x_{2n}. \]

when the following variable order is applied;

\[ x_1 < x_3 < x_5 < \ldots < x_{2n-1} < x_2 < x_4 < \ldots < x_{2n} \]

The ROBDD has size of \(2^{n+1}\) nodes as shown in Figure 28.

![Figure 28: Sub-optimal ROBDD. Source [20]](image)

Using a different variable order,

\[ x_1 < x_2 < x_3 < \ldots < x_{2n-1} < x_{2n} \]
The ROBDD size comes down to \(2n+2\) nodes as shown in Figure 29.

![Figure 29: Optimal ROBDD for the given function using good variable ordering. Source [20]](image)

Several Heuristics are available that can be used to obtain optimal variable order for given function. This is a NP hard problem. Commercial BDD packages like CUDD [22] and Buddy [23] have in built modules that utilize one or more such heuristics to find the optimal variable order and reduce a given BDD to its ROBDD.

### 3.4 Mapping SDMLp to BDD

A node of a BDD can be represented as a 2:1 MUX. Figure 30 represents a BDD representation of a 2:1 MUX [24].

![Figure 30: BDD for 2:1 MUX, \(F = S'A + SB\). Source [24]](image)
From Chapter 1, we can use SDMLp cell (2:1 MUX designed using dynamic signaling and complementary pass transistors logic) to program it to perform any 2 input logic function.

Logic synthesis based on BDD enables the use of third signal (S) of SDMLp as a different input that can be mapped to a BDD using a 1:1 mapping. Thus, there are 3 different types of nodes as explained below;

Node with zero out-degree (terminal node), node with out-degree two and with children nodes being terminal (leaf node) and the node with out-degree two with both children being non-terminal (non-terminal node).

BDD representation for the function $F = (X_1 + X_2)X_3'$ and variable order $X_1 < X_2 < X_3$ is shown below.

![BDD Diagram](image)

Figure 31: ROBDD to explain different Nodes. Source [24]

SDMLp can be mapped from a given BDD by following the following steps:

- Leaf nodes replaced by primary inputs and their edges replace by their respective input variables.
- Every node is replaced by SDMLp cell. The input variable of the replacing
node is given as the select signal (S) of a SDMLp and the two inputs are the low and high edges of the replacing node.

![Diagram](image)

Figure 32: Mapping SDMLp to a ROBDD. Source [24]

Figure 32 shows the two-step process of mapping SDMLp to a ROBDD including complimentary signals. One of the Primary input X3’ replaces the leaf node 3. Input X2 replaces the lower edge if the leaf node.

The CA based FPGA architecture was proposed in Chapter 2. The process of mapping SDMLp cells from a BDD was discussed in the previous section. Now, the complete flow for the FPGA is discussed using ROBDD synthesis and Tree Restructuring Approach for mapping to a FPGA architecture.

The multiplexer property of the SDMLp was realized using Binary Decision Diagram (BDD) synthesis flow, which is proposed in this thesis.

Several synthesis techniques based on BDDs have been proposed in literature, however, they have not been used extensively for synthesis of secured libraries.

In this thesis, a new design flow has been proposed for Fixed Array and Logic Array architecture synthesis, using BDDs for the secure SDMLp library.
3.5 Proposed Synthesis Flow

The benefits of using the multiplexer property of the SDMLp logic style are:

- Single Cell Library Realization
- Cell Similarity and Uniformity (essential for DPA resistant designs).

In the proposed synthesis flow, the input is a Boolean Expression. The given function is represented as a ROBDD, using variable order generator (obtained from CUDD package) and Binary Reduction Unit. The output netlist obtained (ROBDD) is then modified using a tree restructuring approach also called Modified BDD structure (MBDD) to realize a CA based FPGA architecture. The obtained netlist is then mapped to SDMLp using one on one mapping and the SDMLp netlist is processed to obtain a secure structural netlist.

In the Fixed Array architecture, the flow generates the magic layout using vias. In the Logic Array architecture, the flow generates the layout and the programmable bits for the synthesized layout.
3.6 BDD Synthesizer Module

The BDD Synthesizer Module [24] has two main components: The Variable Order Generator (VOG) and the BDD Reduction Unit (BRU).

The VOG generates the variable order for an optimal ROBDD implementation. The output netlist obtained from VOG is given to the BRU.
The CUDD package has “blif2bdd” function [28], which takes in the BLIF file and generates the optimal variable order and the corresponding ROBDD as the output.

BDD Reduction Unit (BRU): The BRU [29][30] forms the other major part of BDD synthesizer module. The input netlists to a BRU are Boolean Expression and Variable Order. It generates the structural netlist from the given inputs.

BDD Reducer: The BDD reducer program co-factors every minterm of the given Boolean expression and generates ROBDD by merging isomorphic graphs, removing the node with isomorphic children nodes and merging complementary nodes. The modules “build_bdd” [19] and “create_bdd” constitute majority of the BDD Reducer module [29]. The algorithms used for these modules are shown in Figures 35 and 36 respectively.

\[
\text{Build\_BDD}(F, i) \quad \text{// Function } F, \text{ index } i \text{ of variable order } [1...n]
\]

\[
\text{Begin}
\]

\[
\text{If (i greater than number of variables) then}
\]

\[
\text{If (} F \text{ is False) then return Zero terminal node;}
\]

\[
\text{Else return One terminal node;}
\]

\[
\text{Else}
\]

\[
\text{Left Child -> Build\_BDD}(F[x_i'], i+1); \quad \text{// Find negative cofactor of } F \text{ with respect to } x_i
\]

\[
\text{Right Child -> Build\_BDD}(F[x_i], i+1); \quad \text{// Find positive cofactor of } F \text{ with respect to } x_i
\]

\[
\text{Return Create\_BDD (Left Child, Right Child, i)};
\]

\[
\text{End}
\]

Figure 35: Co-Factoring Algorithm. Source [24]

Figure 34: BLIF format of a Boolean Function. Source [24]
The BRU returns the root node of the created ROBDD and invokes netlist generator, which performs a Breadth First Traversal and prints out a synthesizable HDL format netlist.

### 3.7 Modified BDD Structure (MBDD) Module

A new method to obtain Modified BDD Structure using Squashed Binary Tree (MSBT) [31] representation is discussed. MSBT approach makes it possible to restructure a BDD into a rectangular form that resembles CA type FPGA architecture. Mapping SBT to a CA type FPGA architecture is a straightforward process as each node is placed in one column of the target array and the necessary connections are made.

Squashed Binary Tree (SBT): A SBT [31] is formed by projecting binary tree to its leaves. The algorithm for SBT is as shown.

- Start from root, and project the nodes to their leftmost descendants.
- Traverse the tree bottom-up direction
- If the node has two children, repeat process starting with child node, which was not projected earlier.

Modified Squashed Binary Tree (MSBT): MSBT is obtained by compacting the SBT. The nodes of the Binary Tree are projected onto the leaves in the Depth-First Manner. The MSBT \(T_{b}(V_{b},E_{b})\) consists of a set of Vertices \(V_{b}\) and set of directed edges \(E_{b}\).
\[ V_b = \{ v_b \mid v_b \text{ represents a set of nodes of a tree, projected onto the same vertex of } T_b(V_b, E_b) \} \]

\[ E_b = \{ e_b \mid e_b \text{ represents a directed edge from } v_{bi} \text{ to } v_{bj} \text{ if any of the nodes of the tree which were collapsed to nodes } v_{bj} \text{ was connected to any of the nodes of the tree which were collapsed to the vertex } v_{bj} \} \]

The vertex of the MSBT are labeled as \( v_{b1}, v_{b2}, \ldots, v_{bn} \) where \( n \) is the number of leaf nodes of the original BDD. Each Vertex \( v_{bi} \) is a set of nodes \((v_1, v_2, \ldots, v_m)\) of the BDD which were collapsed to that vertex. An edge exists between \( v_{bi}, v_{bj} \) if there exists an edge \( e_{kl} \) between \( v_k \) and \( v_l \), where \( v_k \) is the node that was collapsed to vertex \( v_{bi} \) and \( v_l \) is the node which is collapsed to the vertex \( v_{bj} \).

Figure 37 shows the BDD with its corresponding SBT and MSBT representation.

![Figure 37: BDD and corresponding SBT and MSBT. Source [31]](image)

The MBDD module proposed in this work (written in C++), takes the ROBDD netlist obtained from the BDD synthesizer module and converts it to a MSBT representation and the corresponding output netlist is sent to the Technology Mapping Module.

### 3.8 Technology Mapping Module

Technology mapping for the obtained netlist into the CA based FPGA architecture is implemented using MSBT mapping algorithm and is discussed in this section. Each Vertex of the MSBT is implemented in one column of the CA based FPGA array. The number of vertices of the MSBT is equal to the number of columns of the FPGA array. The maximum number of nodes of the BDD, which are projected into one vertex of MSBT determines the number of rows required.

The algorithm used in this work for mapping the MSBT to FPGA architecture is described below.
Algorithm:

- For all vertices $v_{b_1}$ of MSBT, place all nodes of $v_i$ belonging to $v_{b_1}$ in column one.

- Place nodes belonging to node $v_{b_2}$ in column 2, starting with Row(j) defined by the edge between $v_{b_1}$ and $v_{b_2}$.

- If node $v_k$ belonging to $v_{b_1}$ is connected to node $v_l$ belonging to $v_{b_2}$ then $v_l$ is placed in the same row(j) as node $v_k$.

- Repeat the process till all vertices are placed.

This algorithm ensures that all the nodes with vertex(0) child node are placed in the same column and vertex(1) child node are placed in the same row.

Figures 38 shows the SBT for a BDD. The obtained MSBT from the given SBT is shown in figure 39. Figure 40 and 41 show the mapping of SBT and MSBT into the FPGA array. The nodes with dummy nodes (R) can be implemented using SDMLp PLB designed to work as a buffer.

![Figure 38: Squashed Binary Tree (SBT). Source [31]](image1)

![Figure 39: Modified Squashed Binary Tree (MSBT). Source [31]](image2)
Figure 40: Mapping SBT into FPGA Array. Source [31]

Figure 41: Mapping MSBT into FPGA Array. Source [31]
MSBT mapping results in smaller number of columns and the routing cells (nodes) needed for layout implementation than compared to SBT.

The technology mapping module (implemented in C++), has two inputs. The netlist obtained after implementing the algorithm constitutes one input. The other input is the SDMLp library netlist, which has cell information and the routing signals information. The output is a netlist (.mag) file and also a bit-stream file that can be fed as an input to the proposed FPGA architecture to realize the given input function. The 1-bit of the bit-stream file turns on the DFF that denotes the connection is made, and the 0-bit denotes DFF is off. It is a 260000 bits long file, that is fed to the DFF scan chain to program the 100X100 PLB’s CA based Logic Array architecture.

For Fixed Array architecture, the output is a netlist (.mag) file with the interconnects already made using vias.

**3.9 Summary**

In this Chapter, the design flow for the CA based Fixed Array and Logic Array architecture using SDMLp cells is proposed. The packages used for BDD synthesizer module was presented. The proposed MBDD synthesis and technology mapping to the FPGA is presented. The output of the flow gives a MAGIC (.mag) netlist and also a bit-stream that are used as programmable bits input to the proposed FPGA architecture.

In the next Chapter, the experimental setup for the proposed design flow and the evaluated results are presented.
Chapter 4

Experimental Results and Analysis

The design methodology for CA based FPGA architecture using Binary Decision Diagram synthesis was proposed in the previous chapter. In this Chapter, the design metrics of SDMLp cell (Variance and Current Deviation), which is the PLB of the proposed FPGA architecture is evaluated and compared with other logics like WDDL. Next, we implement cryptographic algorithm (DES) [10] using the proposed design flow and evaluate the metric for security (maximum instantaneous current variance).

4.1 Experimental Setup

Experiments were performed using 90nm library obtained from Synopsys tools. Magnitude of 5V was used as supply voltage for the setup. The SDMLp cell and WDDL cell (with MUX implementation) was done using MAGIC editor. In the first part of experimental setup, the Variation and Standard Deviation of SDMLp and WDDL cells are evaluated using SPICE simulations. The implementation was done for a random 4 input function with different Variable Orders.

In the second part of experimental setup, we perform DPA attack using DES algorithm, and evaluate and analyze the current traces [using the flow provided by Antarpreet Singh]. We calculate the Correlation Coefficient for 1000 vectors and plot the graphs for Correlation Coefficient vs. Total vectors and vs. Key Guesses.

The Flow for Experimental Setup is shown in Figure 42.

4.2 Design Metrics Evaluation for Proposed Flow

In this section we evaluate the design metrics (power, current variance and current standard deviation) for SDMLp and WDDL cells. A random 4 input 1 output function is taken and parsed into the proposed flow. The output MAGIC (.mag) file obtained is converted to SPICE file. The results are calculated using HSPICE.

The library used is Synopsys 60nm technology, and the supply voltage used is 5V.
Figure 42: Experimental Setup for the Proposed Flow.

The following figures are the MAGIC layout outputs obtained from post-synthesis of SDMLp and WDDL architectures.
Figure 43: WDDL Layout with Variable Order A, B, C, D.

Figure 44: WDDL Layout with Variable Order D, B, C, A.
Figure 45: SDMLp Layout with Variable Order A, B, C, D.

Figure 46: SDMLp Layout with Variable Order D, B, C, A.
The following table shows the Area and Power comparisons of SDMLp and WDDL logics.

<table>
<thead>
<tr>
<th>Design Metric</th>
<th>WDDL</th>
<th>SDMLp</th>
<th>% Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>VOQ</td>
<td>A, B, C, D</td>
<td>D, B, C, A</td>
<td></td>
</tr>
<tr>
<td>Area (um²)</td>
<td>154.37</td>
<td>158.25</td>
<td>132.54</td>
</tr>
<tr>
<td>No. of Cells</td>
<td>15</td>
<td>12</td>
<td>17</td>
</tr>
<tr>
<td>Total Power (uw)</td>
<td>482.2</td>
<td>456.5</td>
<td>376.9</td>
</tr>
</tbody>
</table>

Figures 47 and 48 shows the graphical representation of area and power for different logics.

![Figure 47: Area in um² for SDMLp and WDDL Logics.](image-url)
Next we evaluate the security metrics for the above design. Current Variance and Current Standard Deviation are best known security metrics. The following table compares the two metrics’ for SDMLp and WDDL logic families.

The following table shows the Current Variance and Current Standard Deviation for the SDMLp and WDDL logics.

<table>
<thead>
<tr>
<th>Security Metric</th>
<th>WDDL</th>
<th>SDMLp</th>
</tr>
</thead>
<tbody>
<tr>
<td>VOG</td>
<td>A, B, C, D</td>
<td>D, B, C, A</td>
</tr>
<tr>
<td>Current Variance (x10^7)</td>
<td>167.29</td>
<td>162.43</td>
</tr>
<tr>
<td>Current Standard Deviation (x10^-4)</td>
<td>40.91</td>
<td>39.12</td>
</tr>
</tbody>
</table>

Figure 49 shows the graphical representation of Current Variance and Standard Deviation for both logics for the given function.
From the above results, it can be inferred that SDMLp logic family contributed for more savings in Power and Area (around 20-30%) as compared to WDDL logic family. Also, the SDMLp cell is evaluated on the basis of Current Variance and Standard Deviation (security metrics) and is shown to have better resistance to DPA attacks than WDDL logic.

4.3 Design Metrics for the Programmed FPGA Architecture

In this section we compare the Area, Delay and Power characteristics of a standard programmable PLB of the proposed architecture flow to that of a normal SDMLp based MUX. The design for a single PLB including its routing structure (shown in Figure 50) was done using MAGIC layout tool. The design metrics for the extracted model of the PLB was evaluated using HSPICE simulation tool. The following figure shows a screenshot of a single PLB with its routing structure, that was simulated using MAGIC layout editor.
The following table gives the Area, Power and Delay performance of the PLB structure with scan chains when compared to that of a normal SDMLp Fixed Array design (SDMLp – FA).

<table>
<thead>
<tr>
<th>DESIGN METRICS</th>
<th>SDMLp - FA</th>
<th>PLB</th>
</tr>
</thead>
<tbody>
<tr>
<td>AREA ($\text{u}^2$)</td>
<td>4308.48</td>
<td>99090</td>
</tr>
<tr>
<td>DELAY</td>
<td>1.1454n</td>
<td>1.9244n</td>
</tr>
<tr>
<td>POWER (uW)</td>
<td>3.0259</td>
<td>9.6463</td>
</tr>
<tr>
<td>---------------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>VARIANCE (x10^6)</td>
<td>75.0719</td>
<td>168.775</td>
</tr>
<tr>
<td>STANDARD DEVIATION</td>
<td>0.00086644</td>
<td>0.00129914</td>
</tr>
</tbody>
</table>

The delay in the above table shows the worst case delay. The following table gives the delays of the outputs with respect to the input signals (Inp0 to Inp7).

<table>
<thead>
<tr>
<th>DELAY</th>
<th>SDMLp - FA (ns)</th>
<th>PLB (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inp0</td>
<td>0.402</td>
<td>1.9027</td>
</tr>
<tr>
<td>Inp1</td>
<td>0.488</td>
<td>1.9064</td>
</tr>
<tr>
<td>Inp2</td>
<td>0.572</td>
<td>1.9087</td>
</tr>
<tr>
<td>Inp3</td>
<td>0.648</td>
<td>1.9102</td>
</tr>
<tr>
<td>Inp4</td>
<td>0.703</td>
<td>1.9136</td>
</tr>
<tr>
<td>Inp5</td>
<td>0.796</td>
<td>1.9179</td>
</tr>
<tr>
<td>Inp6</td>
<td>0.858</td>
<td>1.9208</td>
</tr>
<tr>
<td>Inp7</td>
<td>0.926</td>
<td>1.9244</td>
</tr>
</tbody>
</table>

A specific set of inputs was selected control the delay of the cell, making it worthwhile to study the affects of timing based attack.

The following waveforms (simulated using HSPICE and CSCOPE) show the functioning of the normal SDMLp cell as a MUX and the PLB designed using SDMLp functioning as a MUX. The delay in both cases has been evaluated and captured.
Figure 51: SDMLp cell used as MUX (SDMLp – FA)

Figure 52: Delay of SDMLp cell implemented as a MUX (SDMLp – FA)
Figure 53: PLB for proposed FPGA architecture (SDMLp – PLB)

Figure 54: Delay of single PLB of proposed FPGA architecture (SDMLp – PLB)
The following table shows the scaling of the FPGA architecture based on the number of PLB’s.

<table>
<thead>
<tr>
<th>PLB SIZE</th>
<th>DIMENSIONS</th>
<th>AREA ($u^2$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1x1</td>
<td>734x1500</td>
<td>99090</td>
</tr>
<tr>
<td>30x30</td>
<td>22020x4500</td>
<td>89181000</td>
</tr>
<tr>
<td>50x50</td>
<td>36700x7500</td>
<td>247725000</td>
</tr>
<tr>
<td>100x100</td>
<td>73400x15000</td>
<td>990900000</td>
</tr>
</tbody>
</table>

The following figure 55 shows the MAGIC layouts for the proposed FPGA architecture for PLB sizes 30x30, 50x50 and 100x100 and figure 56 shows the magnified cross-section of the PLB interconnections.
4.4 Security Metrics Evaluation for Proposed Flow

In this section, we evaluate the security metrics of the proposed flow by performing a DPA attack using DES algorithm. We measure and evaluate the correlation coefficient with respect to the key guess and also with total number of vectors.

Data Encryption Standard (DES) [10]

DES is the frequently used algorithm commercially. The 64-bit plain text is divided into $L_0$ and $R_0$ sub blocks, the values of which are determined by the following equations,

\[ L_i = R_{i-1} \]

\[ R_i = L_i \oplus f(R_{i-1}, K_i) \]
Figures 57 and 58 show the Fiestel Function and the DES Algorithm Implementation respectively.

Figure 57: DES Fiestel Function. Source [10]
Figure 58: DES Algorithm Implementation Overview. Source [10]

Figure 59 shows the final layout of all the 8 S-Boxes in the proposed CA based FPGA architecture. Figure 60 shows each S-Box separately for SDMLp – FA architecture.
The following table shows the Area and Power utilization for each S-Box for SDMLp
– FA architecture.

<table>
<thead>
<tr>
<th>COMPONENT</th>
<th>AREA ($x10^6 \lambda^2$)</th>
<th>POWER ($10^{-5}$ W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S-BOX 1</td>
<td>17.27</td>
<td>13.24</td>
</tr>
<tr>
<td>S-BOX 2</td>
<td>16.94</td>
<td>12.86</td>
</tr>
<tr>
<td>S-BOX 3</td>
<td>15.01</td>
<td>14.39</td>
</tr>
<tr>
<td>S-BOX 4</td>
<td>15.88</td>
<td>11.62</td>
</tr>
<tr>
<td>S-BOX 5</td>
<td>16.93</td>
<td>13.77</td>
</tr>
<tr>
<td>S-BOX 6</td>
<td>18.8</td>
<td>15.51</td>
</tr>
<tr>
<td>S-BOX 7</td>
<td>14.65</td>
<td>11.29</td>
</tr>
<tr>
<td>S-BOX 8</td>
<td>18.26</td>
<td>12.23</td>
</tr>
</tbody>
</table>

Figure 61 shows the graphical representation of Area and Power of SDMLp logic DES S-Boxes using the proposed flow for SDMLp – FA architecture.

Figure 61: SDMLp DES S-Box Area and Power for SDMLp – FA.
The next table shows the current variance and standard deviation for each of the 8 S-Boxes for SDMLp – FA architecture.

<table>
<thead>
<tr>
<th>COMPONENT</th>
<th>Variance (x10⁻⁷)</th>
<th>Standard Deviation (10⁻⁴)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S-BOX 1</td>
<td>64.96</td>
<td>2.55</td>
</tr>
<tr>
<td>S-BOX 2</td>
<td>60.9</td>
<td>2.47</td>
</tr>
<tr>
<td>S-BOX 3</td>
<td>61.03</td>
<td>2.51</td>
</tr>
<tr>
<td>S-BOX 4</td>
<td>62.65</td>
<td>2.49</td>
</tr>
<tr>
<td>S-BOX 5</td>
<td>62.3</td>
<td>2.48</td>
</tr>
<tr>
<td>S-BOX 6</td>
<td>61.82</td>
<td>2.5</td>
</tr>
<tr>
<td>S-BOX 7</td>
<td>60.43</td>
<td>2.45</td>
</tr>
<tr>
<td>S-BOX 8</td>
<td>60.66</td>
<td>2.46</td>
</tr>
</tbody>
</table>

Figure 62 shows the graphical representation of Current Variance and Standard Deviation of SDMLp logic using the proposed flow for SDMLp – FA architecture.

The following figure 63 shows the final layout of all the 8 S-Boxes of the proposed...
FPGA architecture, using SDMLp – PLB.

The following table gives the area and dimensions of each S-Box implemented using SDMLp – PLB architecture.
<table>
<thead>
<tr>
<th>S-BOX</th>
<th>DIMENSIONS</th>
<th>AREA (u²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S-BOX 1</td>
<td>7x51</td>
<td>35375130</td>
</tr>
<tr>
<td>S-BOX 2</td>
<td>7x50</td>
<td>34681500</td>
</tr>
<tr>
<td>S-BOX 3</td>
<td>6x52</td>
<td>30916080</td>
</tr>
<tr>
<td>S-BOX 4</td>
<td>7x47</td>
<td>32600610</td>
</tr>
<tr>
<td>S-BOX 5</td>
<td>7x51</td>
<td>35375130</td>
</tr>
<tr>
<td>S-BOX 6</td>
<td>7x56</td>
<td>38843280</td>
</tr>
<tr>
<td>S-BOX 7</td>
<td>6x51</td>
<td>30321540</td>
</tr>
<tr>
<td>S-BOX 8</td>
<td>7x54</td>
<td>37556020</td>
</tr>
</tbody>
</table>

In the next experiment, we use the proposed flow and implement a 5-Bit and 8-Bit parity checkers using xor-tree implementations. The following figures (Figure 64 and Figure 65) show the gate level diagrams for the parity checkers used.

Figure 64: 5-Bit Parity Checker.
The following figures (Figure 66 and Figure 67) show the MAGIC layouts for 5-Bit Parity Checker function and 8-Bit Parity Checker functions using SDMLp based PLB are generated using the proposed Flow.
The following figures show the IRSIM simulation results for the 5-Bit Parity Checker and 8-Bit Parity Checker. The delay associated with 5-Bit Parity Checker and 8-Bit Parity Checker circuits implemented using the proposed flow using SDMLp based PLB FPGA architecture.

The delay for 5-Bit parity checker using SDMLp-PLB implementation is 447.1ns and for 8-Bit parity checker using SDMLp-PLB implementation using the proposed flow is 572.11ns.
Figure 68: 5-Bit Parity Checker using SDMLp – PLB waveform.

Figure 69: 5-Bit Parity Checker Delay using SDMLp – PLB.
Figure 70: 8-Bit Parity Checker using SDMLp – PLB.

Figure 71: 8-Bit Parity Checker Delay using SDMLp – PLB.
In the next experiment, we perform the DPA attack on DES and plot the waveforms for Correlation Coefficient vs. number of vectors and key guesses.

### 4.5 DPA Attack on DES

Here, Differential Power Analysis (DPA) [15] attack is performed on a single round of DES sub-circuit. This sub-circuit, shown in figure 72 is proven to be sufficient to analyze the security metric of the DES chip. DES S-Box 8 was implemented and it’s functional block is as shown below. The DPA attack (using the platform developed by Antarpreeth Singh, DDEL, University of Cincinnati) is performed on the SDMLp logic obtained using the proposed flow. The secret key for data encryption is chosen to be 17.

![Figure 72: DES Circuit – S-Box 8. Source [15]](image)

Figure 73 shows the waveform plot of Correlation Coefficient vs. the total number of Vectors used during the DPA attack for SDMLp logic. The line in bold represents the Key (17) that was used for the attack. Thus, it can be seen in the plot that the Correlation Coefficient for the original key is well hidden among other correlation values used to perform the DPA attack and hence improving the resistance against the attack.

The Correlation Coefficient vs. Key Guesses for 1000 random vectors for SDMLp – FA architecture is as shown below. Since there is no distinct peak with single key guess, identifying the original key becomes difficult, proving the logic is DPA resistant.

Figure 75 shows the waveform plot of Correlation Coefficient vs. Number of Vectors for SDMLP – PLB based architecture, and Figure 76 shows the Correlation
Coefficient vs. Key Guesses for SDMLp – PLB architecture.

Figure 73: Correlation-Coefficient vs. Total Vectors for SDMLp – FA.

Figure 74: Correlation-Coefficient vs. Key Guesses for SDMLp – FA.
Figure 75: Correlation-Coefficient vs. Total Number for SDMLp – PLB.

Figure 76: Correlation-Coefficient vs. Key Guesses for SDMLp – PLB.
4.6 Conclusions and Analysis

The proposed SDMLp logic flow for CA based FPGA synthesis shows significant improvements in Area and Power (Design Metrics) as compared to WDDL flow. It can be seen from the experiment conducted by using a random 4 input function, that, the Area improvement over WDDL logic is about 20% and Power consumption improvement over WDDL logic is about 24%. The SDMLp logic for the proposed flow shows encouraging numbers in terms of area and power utilizations when DES cryptographic circuit is implemented. In the final experiment, DPA attack on a DES circuit using SDMLp logic for the proposed flow is performed for both SDMLp – FA architecture and SDMLp – PLB architecture. The results show that the Correlation Coefficient is well hidden among other values, and also there are multiple peaks for key guesses, thereby, showing that this circuit implementation is DPA attack resistant.
Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this work, we present a top-down method for DPA resistant CA based FPGA architecture using SDMLp logic. The performance of SDMLp logic over other existing DPA resistant logics like WDDL in FPGAs was compared and evaluated.

A new BDD based synthesis flow (Tree Restructuring using MSBT) to realize CA based FPGA architecture was proposed. DPA attack on this flow using cryptograhic circuit (DES) was performed and the design metrics (Area, Power, Current Variance and Standard Deviation) were evaluated and shown.

It is shown that the area reduction in case of SDMLp logic was about 20% over WDDL logic, and the improvement in terms of power is about 24% over WDDL logic.

A DPA attack was performed on a DES S-Box circuit for the proposed flow using SDMLp logic and the results show that the secret key could not be retrieved, showing that the design is secure against the DPA attack.

5.2 Future Work

There are few future works that can be derived from this thesis work.

- The Proposed Flow for secure CA based FPGA architecture was done using BDD synthesis using SDMLp cell as a MUX. This was a two input MUX cell and hence the PLB used in this architecture was two-input MUX, restricting the size of the PLB and thus increasing the cap on the routing used and area of the entire FPGA in general.

- BDD synthesis using SDMLp also meant only combinatorial circuits could be designed. This restricted the usage of sequential circuits, and hence the FPGA architecture using the proposed flow worked only for combinational logic.

- The size of the FPGA architecture was restricted to 100x100 PLB, thus restricting the flow to work for a maximum of 8-input and 8-output functions.
• The proposed flow algorithm can be improved to decrease the total number of cells than currently used thereby increasing the efficiency of FPGA and also reduce power consumption.

• The algorithm used for routing the FPGA can be improved, thereby, removing redundant routing wires and improving the overall FPGA area and also the DFF’s needed for switching in the routing circuit.
Bibliography


[23] “BUDDY – A BDD Package”. By Jorn Lind Nielsen


[26] Slideshare on ACTEL FPGA’s. Available at: http://www.slideshare.net/anishgupta94801/actel-fpga


[32] Borowczak, Mike, University of Cincinnati, Cincinnati, 2013, <https://scholar.google.com/citations?user=h0M0TmsAAAAJ&hl=en>


