Architectures des FPGAs Asynchrones pour les Applications Cryptographiques

Sumanta Chaudhuri

École Doctorale d'Informatique, Télécommunications et Électronique de Paris

Thèse

présentée pour obtenir le grade de docteur de l'Ecole Nationale Supérieure des Télécommunications

Spécialité : Electronique et Communications

Sumanta Chaudhuri

Asynchronous FPGA Architectures for Cryptographic Applications.

15 Mai, 2009
Le Jury

Pr. Claude Chappert (IEF, Orsay)	*Président*
Pr. Lionel Torres (LIRMM, Montpellier)	*Rapporteurs*
Pr. Habib Mehrez (LIP6, Paris)	
Dr. Laurent Fesquet (TIMA, Grenoble)	*Examinateurs*
Dr. Jean-Michel Vuillamy (Altera, France)	*Invité*
Dr. Sylvain Guilley (ENST, Paris)	*Directeurs de thèse*
Pr. Jean-Luc Danger (ENST, Paris)	

Dedicated to Malabika & Syamal Kumar Chaudhuri who passed onto me their "Joie de Vivre".

Dédié à Malabika & Syamal Kumar Chaudhuri qui m'ont passé leurs joie de vivre.

Remerciements

Tout d'abord, je voudrais remercier Jean-Luc Danger pour avoir encadré mon travail depuis le Master et pendant la thèse. Je lui dois toute ma reconnaissance pour m'avoir accueilli dans l'équipe SEN de l'ENST où j'ai pu travailler sur ma thèse et sur bien d'autres sujets passionnants.

Je tiens aussi à remercier tout particulierement Sylvain Guilley pour ses conseils et ses encouragements tout au long de ma thèse. Ses bons réflexes scientifiques, sa force de travail, son enthousiasme et son intrépidité ont toujours provoqué ma grande admiration.

Je voudrais également remercier Philippe Hoogvorst et Yves Mathieu pour les nombreuses discussions et débats enrichissants, Florent Flament, Laurent Sauvage et Tarik Graba qui étaient d'excellents collègues, et Fréderic Pauget qui était toujours prêt á aider.

Un grand merci à Karim Ben Kalia pour la satisfaction d'aller jusqu'à l'implémentation physique qui n'aurait pas était possible sans lui. Karim a assuré tous les détails pratiques des expériences mentionnées dans ce travail de thèse.

Je voudrais également exprimer ma gratitude á Chantal Cadiat, Marie Baquero, Cécile Guyon et Florence Besnard pour m'avoir épargné de nombreux détails administratifs.

Je remercie tous mes collegues d'autres institutions avec qui nous avons travaillé, notamment Hayder Mrabet, Zied Marrakchi, Hussein Parvez du LIP6 pour des échanges et collaborations très fructeux, Taha Beyrouthy, Laurent Fesquet et Marc Renaudin du laboratoire TIMA á Grenoble, avec qui nous avons achevé le projet SAFE. Je remercie tous mes collegues d'IEF á Orsay, notamment Djafar Chabi, Annerose Helmer,Sanghwan Park, Capucine Burrowes, Laurence Bianchini, Jean-Marie Retrouvey, Pierrick Balestriére et Joo-Von Kim qui m'ont acceuilli chaleuresement dans l'equipe Nanospintronics.

Je voudrais exprimer ma gratitude à tous les membres de mon jury de thèse et notamment à mes rapporteurs de thèse Lionel Torres professeur au LIRMM à Montpellier, Habib Mehrez Professeur au LIP6 à Paris, et à Claude Chappert qui a accepté d'être le président du jury malgré son itineraire chargé.

À titre plus personnel, je me souviendrais toujours de mes amis, á télécom, sur Paris et á Orsay. Nous avons vécu beaucoup de moments inoubliables pendant ces quatre années. Je pense notamment à tous les thésards footballeurs et ping-pongeurs de Télécom, aux basketballeurs de l'IEF, á Sami qui m'a initié aux PhDComics, á real slim Chadi (Chadi Jabbour) le Directeur de Foot, aux Chacha (Hasham Ahmed Khushk), Mirani (Farhan Hayder Mirani), Mota (Shivam Bhasin) et Jadugar (khalil laghari) pour nos aventures nocturnes (culinaires et non-culinaires), aux Nidhal(Nidhal Selmane), Youssef(Youssef Souissi), Aziz(Moulay Abdelaziz Elaabid) pour les innombrables pause café, aux Irina(Irina Dorobat), Kapil(Kapil Mahajan), Loukas(Loukas Papaniaou), Vassilis(Vassilis pervezas), Ioannis(Ionnis Dermitzakis), Rafael(Rafael Bauer), et Andreas(Andreas Schjohhaug): les gens indissociables de mon souvenir de Paris.

Enfin merci á Natacha(Natacha Stawiarski), qui était comme une oasis pour moi pendant ces années.

Abstract

In this thesis manuscript, we present reconfigurable architectures and methods to successfully implement countermeasures against physical cryptanalysis of electronic cryptographic devices. Physical cryptanalysis includes both Fault attacks(i.e Injecting Faults in the circuit, and retrieve the key by observing syndromes, also known as active attacks) and Side-Channel Attacks(i.e passive attacks where the information leaked through power consumption or EM radiation Pattern is used to retrieve the key). These attacks are a growing concern for mathematically secure cryptographic algorithms like AES/Triple-DES.

The thesis begins with a state of the art of Asynchronous Circuits, FPGA Architectures [136], Physical Cryptanalysis [142] and its Counter-Measures, to set up the context. Among the counter-measures we highlight two broad classes, namely dynamic and static counter-measures. Dynamic Counter-Measures rely upon reconfiguration(often random) of the IP and incorporated at Run-Time(e.g MDPL, Dynamic Masking, Path Switching Logic etc). Static counter-measures are incorporated at design-time and often relies on power constant dual-rail signaling(e.g WDDL, Back-End Duplication etc). With this state-of-the-art in mind the motivations for an asynchronous reconfigurable architecture is presented. Some salient points of this technology are: Resistance to fault injection, power constant signaling, randomization of switching activity instants etc.

First, we present a Run-Time Reconfigurable architecture [138], intended for the implementation of dynamic counter-measures against both active and passive attacks, and associated reconfiguration schemes. This part includes automation of the CAD flow [137], based on open source FPGA P/R tool VPR(UofT), Details and micrograph of an 8×8 Run-Time Reconfigurable(RTR) FPGA embedded in a SoC [139], Design of Experiments, Test methods and finally the experimental results. The architecture described is a synchronous FPGA with asynchronous capabilities, and methods focus mainly on reconfiguration schemes rather than the type of logic used. However suitability of asynchronous logic for RTR is highlighted.

Secondly we present a multi-style asynchronous FPGA architecture, designed in collaboration with TIMA (CIS Group) under the "SAFE" project, aimed at prototyping of static counter-measures. This chapter includes the automated design flow with an emphasis on balanced interconnect design [141](Twisted-pair Switchbox and Binary-tree connection Box), floorplan details, micrograph, and experimental results [140]. A simple dual-rail routing algorithm based on VPR for homogeneous routing architectures and associated results are presented. This chapter also includes a description of the PLB(Programmable Logic Block) and tech-mapping of various asynchronous styles [143, 135], which is the subject of an ongoing collaborative research between ENST and TIMA.

Finally I conclude this thesis with emphasis on the use of a mix of dynamic and static counter-measures, to increase resistance(MTD: Measurement to Disclosure: a popular metric to rank counter-measures) to physical cryptanalysis, and as a future research direction, I propose the use of auto-decay mechanisms (i.e Electro- migration, NBTI) to limit the number of times a cryptographic device can be used, evidently less than its MTD.

Contents

Résumé de La Thèse en Français

Résumé de La Thèse en Français

La cryptologie est un moyen de protéger la confidentialité, d'assurer l'intégrité, ou d'authentifier un système, tandis que la cryptanalyse est le moyen de retrouver l'information secrète. La plupart des standards cryptographiques sont conçus en tenant compte des méthodes de cryptanalyse mathématiques. Ainsi les algorithmes cryptographiques modernes tels que AES, DES sont presque impossibles à attaquer au niveau mathématiques. Mais dans le contexte actuel, ces algorithmes cryptographiques sont implémentés sur un machine/substrat physique tel que le circuit électronique dans la plupart des cas. La fuite d'information liée aux fonctionnements de ces machines, ou la manipulation de ces machines pour retrouver la clé secrète est alors devenue un moyen puissant de cryptanalyse. Ces attaques sont connues sous le nom d'attaque par canaux cachés soit "Side-Channel Attacks" en anglais. Les exemples célèbres de ce genre d'attaques dans l'histoire sont le programme "Tempest" aux Etats-Unis ou encore l'attaque par le contenu d'un écran a partir des réflexions dans les yeux [130].

Ce travail de thèse tente de trouver une réponse aux questions suivantes :

- Existe il une architecture dont la fuite d'information ne permet pas à l'attaquant de retrouver la clé plus vite que le cas ou il n'y aurait pas de fuite ?
- Jusqu'à quel point ces fuites sont-elles tolérables, et comment peut-on maximiser l'utilisation de ces machines sans compromettre leur secret ?
- Quelles sont les métriques pour déterminer la vulnérabilité des circuits électroniques face à une multitude de méthodes d'attaques ?

Récemment d'innombrables chercheurs ont posé ces questions pour les différentes catégories de circuits électroniques. Dans ce travail de thèse nous restreignons notre espace de recherche parmi les circuits de type "FPGA" et de type "asynchrone". L'atout principal des circuits FPGA est leur reconfigurabilité, qui peut être utilisée pour adapter l'algorithme face à une attaque. Les circuits asynchrones ont de bonnes propriétés telles que la tolérance aux fautes, la décorrélation de la consommation, qui sont utiles contre les attaques par canaux cachés.

Cryptanalyse Physique

Dans ce chapitre nous présentons un état de l'art pour la cryptanalyse physique et la cryptographie en général, pour mieux comprendre les enjeux et l'importance de ces attaques.

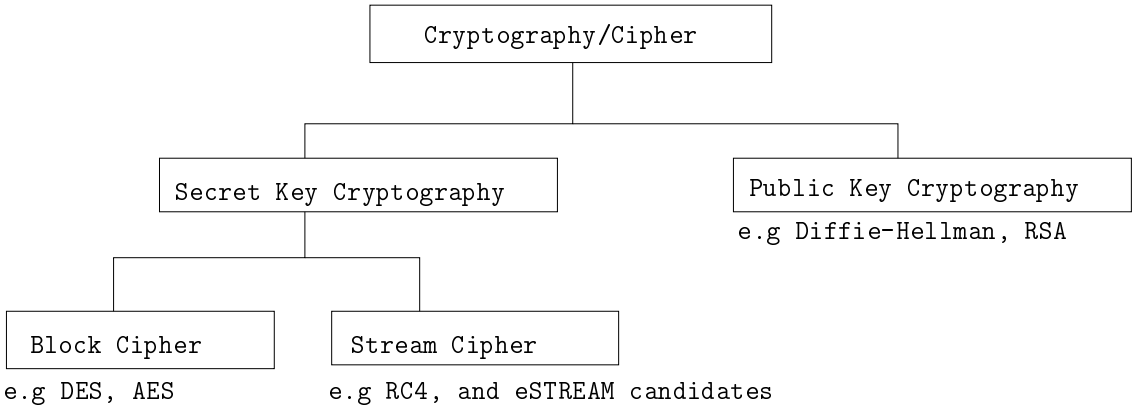


Figure 1: La Classification des chiffrements.

Cryptographie

La cryptographie moderne consiste en deux grandes catégories: Cryptographie à clé secrète et Cryptographie à clé publique (voir fig. 1). Dans la cryptographie à clé secrète, seul l'émetteur et le destinataire connaissent la clé secrète. Cette catégorie peut encore être divisée en deux sous-catégories, chiffrement en blocs et chiffrement en flot. Les standards cryptographiques populaires tels que AES, DES sont des exemples de chiffrement en blocs. Pour un chiffrement en blocs la donnée en série est d'abord convertie en un bloc de taille fixe, et ensuite ces blocs sont encryptés avec le même clé. Quant au chiffrement en flot chaque bit de message subit une transformation variable selon les bits de la clé. Un exemple de Stream cipher est RC4. Dans la cryptographie à clé publique, l'émetteur encrypte le message avec la clé publique du destinataire, et le destinataire décrypte ce message avec sa clé privée. RSA est une implémentation de la cryptographie à clé publique qui à connu le plus grand succès. RSA(Rivest-Shamir-Adleman) nommé d'après ses inventeurs est basée sur le fait qu'il est facile de trouver un nombre premier de grande taille (100 chiffres) mais factoriser un produit de nombres est extrêmement difficile. Même si RSA est considéré un des standards cryptographiques le plus sécurisé, il reste vulnérable aux attaques par canaux cachées ou attaques en fautes qui utilisent les propriétés physiques et non pas les propriétés mathématiques.

Cryptanalyse

En principe toutes les méthodes qui réduisent le nombre d'essais pour retrouver la clé secrète, comparé à une recherche exhaustive, sont considérées comme des méthodes cryptanalytiques. La cryptanalyse peut être divisée en deux catégories distinctes, mathématique et physique(voir fig. 2). Parmi les méthodes de cryptanalyse mathématique, les plus connues sont la cryptanalyse différentielle (aussi connue sous le nom de T-Attack) et la cryptanalyse linéaire. La cryptanalyse différentielle essaie de trouver une corrélation entre la différence de deux messages d'entrée et la différence de deux messages cryptés. Cette méthode a été publiée pour la première fois par Biham & Shamir [7]. Elle requiert des messages clairs prédéfinis pour cette attaque. La cryptanalyse linéaire a été publiée pour la première fois par Mitsuru Matsui [26]. Cette méthode a besoin de messages clairs connus. Même si ces attaques mathématiques ne sont pas dans le contexte de ce travail de thèse, nous les mentionnons, car elles peuvent être utilisées en conjonction avec les attaques

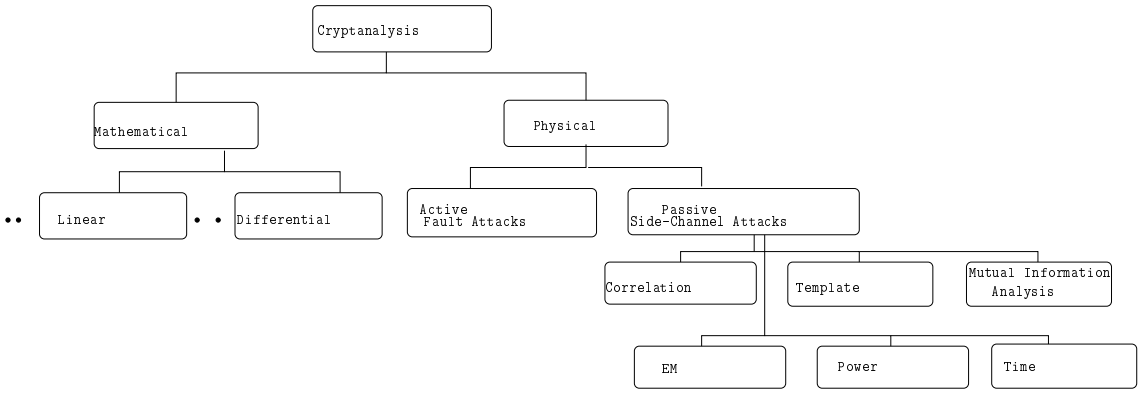


Figure 2: La Classification de la Cryptanalyse.

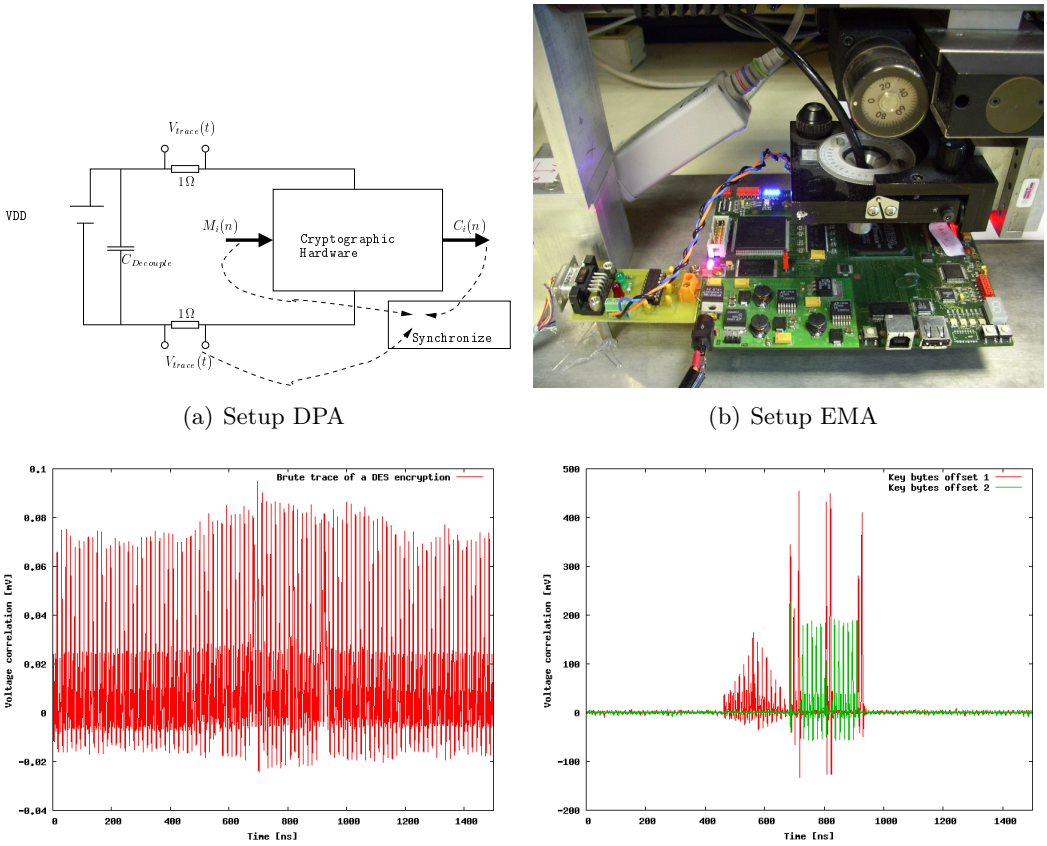
Table 1: SCA: Contre-mesures

	Run Time Reconfiguration	
Static (Dual Rail with Precharge)	Dynamic	
WDDL	RSL	Random Rerouting
Asynchronous Logic (1-out-of-2 4 Phase)	MDPL	Introduction of Random Delays (Asynchronous Logic)

par canaux cachés. La cryptanalyse physique peut être divisée en deux sous catégories, notamment active et passive. Dans le cas d'une attaque active l'attaquant manipule le fonctionnement du circuit, souvent en injectant une faute, pour retrouver la clé secrète. Une attaque passive consiste à collectionner la fuite de l'information et ensuite à corréler avec la valeur de la clé secrète. Ces attaques sont aussi connues sous le nom d'attaques par canaux cachés ou "Side-Channel Attacks". Les attaques par canaux cachés peuvent être divisées en plusieurs catégories selon les méthodes des acquisitions où les méthodes de traitement des données acquises.

La figure 3 explique les méthodes principales de "Side-Channel Attacks". La figure 3(a) explique la méthode d'attaque d'un circuit électronique (ex. Carte à Puce) où la consommation du circuit est enregistrée au moment du chiffrement (une trace de mesure). Ces traces sont ensuite corrélées avec le modèle de consommation (consommation CMOS pour les circuits électroniques numériques), et une partie de la clé est récupérée (celle avec la plus grande corrélation par rapport à la mesure). La figure 3(b) présente les attaques électromagnétiques. Le rayonnement à partir du circuit cryptographique est capté par une Antenne. Les traces de mesures brutes(figure. 3(c)) ne dévoile aucune information et ressemble à du bruit, mais après avoir calculé une fonction de corrélation avec le modèle de consommation, les pics apparaissent pour la vraie valeur de la clé secrète.

Les grandes lignes des contre-mesures sont détaillées dans le tableau 1. Il existe deux différents types de contre-mesures, statique et dynamique.Les contre-mesures statiques ont une consommation de courant ou émission électromagnétique constante, qui ne dépend pas des données. Une catégorie de contre-mesures repose des logiques différentielles à précharge telles que WDDL, ou certains protocoles de la logique Asynchrone. Une autre catégorie



(a) Setup DPA

(b) Setup EMA

(c) Des Traces de Mesures après l'Acquisition

(d) Après corrélation

Figure 3: Attaques par canaux cachés

de contre-mesures "dynamique" est à base de masquage. Celles ci essaient de reproduire une trace de consommation aléatoire et embrouillent l'information présente dans les traces de mesures. Des exemples de contre-mesures dynamiques et faisant appel à des logiques à précharge sont RSL et MDPL.

La table 2 présente le résultat du concours sur les attaques par canaux cachés. Elle montre la facilité avec laquelle on peut défaire un standard cryptographique tel que DES (avec 135 mesures). Cela pourrait sembler inquiétant pour les applications bancaires, mais aussi pour la protection de la propriété intellectuelle, ou encore les applications qui dépendent des cartes à puce telles que la TV à péage.

L'architecture et CAO pour FPGA

Dans ce travail de thèse nous nous intéressons aux circuits électroniques de type FPGA. Un FPGA (Field Programable Gate Arrays) est un circuit ayant une architecture reconfigurable capable d'implémenter la plupart des circuits numériques. Même si le fonctionnement est plus lent que des circuits dédiés, l'avantage des FPGA est surtout son adaptabilité.

La figure 4 présente l'élément logique de base d'un FPGA nommé CLB (Configurable Logic Block). Il est constitué d'une LUT (Look Up Table) capable d'implémenter n'importe

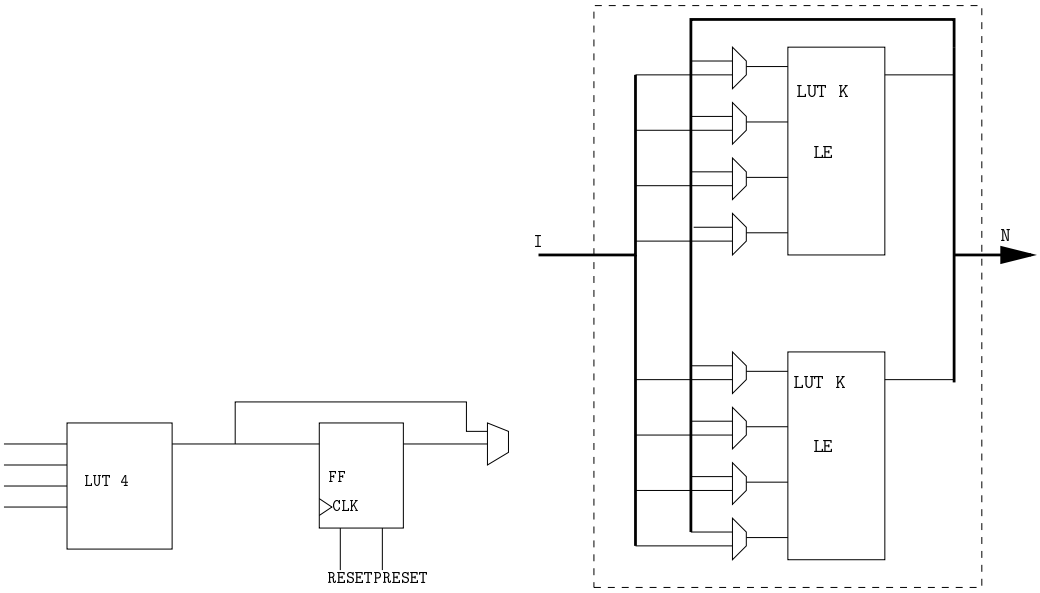
Table 2: Les Resultat de la Concours DPA 2008

Auteur	Methode d'Attaque	Traces
Yongdae KIM, Tohoku university Aoki laboratory	CPA using Pearson's correlation	135
Eloi SANFELIX, Riscure	targets 2 sboxes at a time (12 key bits), using a sum of the correlation above a certain threshold as a criterion	232
Victor LOMNE, LIRMM	enhancement of the CPA (from the work of Thanh-Ha LE et al.) on the 16th round of the DES selecting the good temporal window	310
Benedikt GIERLICHS, KUL	Difference of Means on the last round	329
Sylvain GUILLEY, ENST	multi-bit CPA with fourth-order cumulant preprocessing	569

quelle fonction Booléenne, et une Bascule-D pour les fonctions séquentielles. Des éléments logiques de base peuvent être regroupés dans un cluster (figure. 4(b)). Ces CLBs peuvent communiquer entre eux grâce aux ressources de routage présentes dans le FPGA. Plusieurs types d'architecture sont possibles pour les ressources de routage dans un FPGA, par exemple le routage en mesh (matriciel) et le routage hiérarchique. Dans les FPGA mesh les CLBs sont rangés dans une grille en deux dimensions avec les boîtes de commutations et les boites de connexions.

La figure. 14(a) présente les ressources de routage de base. Les entrées et sorties de CLB sont connectés aux canaux de routage par le biais des boîtes de connexion (connexion box). La matrice de commutation (Switch box) permet aux canaux de routage de se connecter entre eux selon le besoin du circuit implémenté. Chaque connexion dans les ressources de routage se fait par des switchs (commutateurs ou interrupteurs) de type buffer (élément actif) ou une porte de transmission (élément passif) relié à un point mémoire. L'ensemble de ces points mémoires correspond à la configuration(ou bitstream) d'un FPGA. Dans la figure 14(b) nous voyons un exemple de routé d'une sortie de CLB vers une entrée d'un autre CLB en utilisant les ressources de routage d'un FPGA. Les points mémoires reliés sont alors mis à '1' pour construire cette connexion.

Vu l'architecture du FPGA, il est évident que le flot de conception pour implémenter un circuit logique dans un FPGA est considérablement différent d'un circuit conventionnel CMOS. La figure 6 présente le flot de conception simplifié pour les FPGAs. L'avantage de ce flot de conception comparé au flot de conception ASIC est sa rapidité. Pour cette raison les FPGAs sont beaucoup utilisés pour le prototypage. Les contre-mesures face aux attaques par canaux cachés souvent utilisées reposent sur des logiques de type dual-rail équilibré. La grand différence entre des circuits ASIC et FPGA sont les ressources de routage sont utilisent beaucoup de switches pour les FPGAs, ce qui va magnifier les déséquilibres présents entre les connexions dual-rails. L'avantage d'utiliser les FPGA pour les contre-mesures est la reconfigurabilité mais la présence de beaucoup de switches dans le routage est un inconvénient. Dans ce travail de thèse nous adressons aussi ce problème de déséquilibre et nous proposons des solutions.



(a) Element Logique. (b) Cluster Logique.

Figure 4: L'Architecture Logique.

Les Circuits Asynchrones

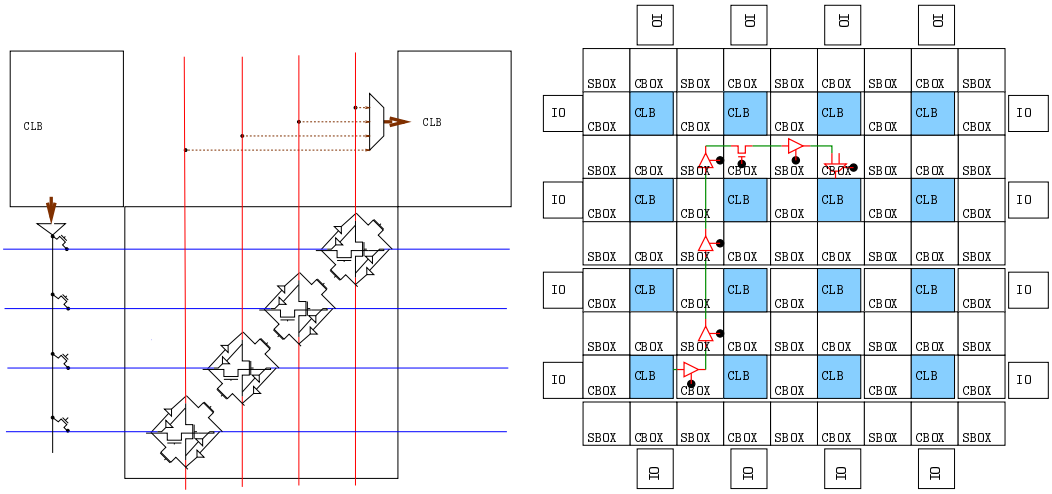
Le chapitre 3 de ce manuscrit de thèse présente les idées clés de la logique asynchrone. Les figures 7(a), 7(b) montrent le protocole asynchrone et les différences avec le protocole synchrone. Dans la logique synchrone, toute communication entre les portes se fait sur front de l'horloge principale. C'est à dire que les données ne sont valides qu'au front montant ou descendant de l'horloge. Quant à la logique asynchrone, l'émetteur envoie la donnée et une requête au destinataire. Une fois que le destinataire a reçu les données, il envoie un acquittement vers l'émetteur, comme expliqué dans la figure 7(b).

Table 3: 1-out-of-2, 4 Phase

Value	DATA0	DATA1
'0'	'1'	'0'
'1'	'0'	'1'
Precharge	'0'	'0'
Forbidden	'1'	'1'

Dans la pratique les données et la requête sont souvent encodés dans le même fil de communication. Un exemple d'encodage est le protocole 1-out-of-2 en 4 Phases (voir la figure. 7(c) et la table. 3). On peut aussi remarquer que la consommation liée à ce protocole est constante et indépendante des données. Celle-ci est un résultat de l'encodage utilisé et n'est pas vrai pour tous les protocoles asynchrones.

Il existe plusieurs types de codage et de protocoles pour les circuits asynchrones, par exemple les codages asynchrones tel que le 1-out-of-N, LEDR (Level Encoded Dual-Rail).



(a) SwitchBox and Connection Box (b) Single Driver Interconnect

Figure 5: Ressources de Routage.

Motivation

Auparavant nous avons mentionné que nous restreignions notre espace de recherche aux circuits de type FPGA asynchrone. Voici les raisons pour lesquelles ce type de circuit a été choisi :

- **Résistance aux attaques en fautes**
- **Absence d'horloge**
- **Consommation indépendante du calcul**
- **Absence de parasite (ou "Glitch")**
- **Reconfigurabilité**

FASE: Architecture d'un FPGA Reconfigurable à Chaud

FASE est le nom que nous avons donné à un FPGA pour implémenter les contre-mesures basées sur la reconfiguration dynamique permettant de contrer les attaques par canaux cachées. La spécialité de FASE est qu'il peut être reconfiguré à chaud, c'est à dire qu'on peut le programmer sans arrêter le fonctionnement du FPGA.

Architecture

FASE est un FPGA embarqué, contrairement à la plupart des FPGAs commerciaux. Le schéma 8 présente une idée du SoC (System on Chip) dans lequel FASE est embarqué. Notre SoC nommé SECMAT utilise le bus VCI pour l'interconnexion entre les différents composants du SoC. Dans la figure. 8 on utilise le microprocesseur FREE6502 qui est le contrôleur principal de ce SoC. Les composants(IP) comme les coprocesseurs cryptographiques sont connectés au bus VCI et contrôlés par le FREE6502. Le FPGA FASE est un de ces

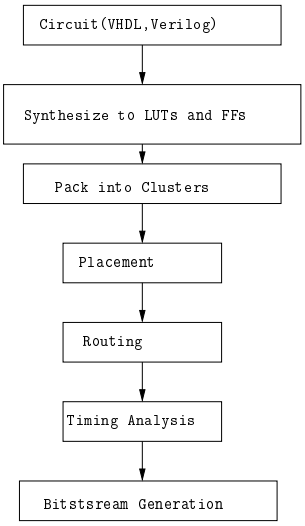
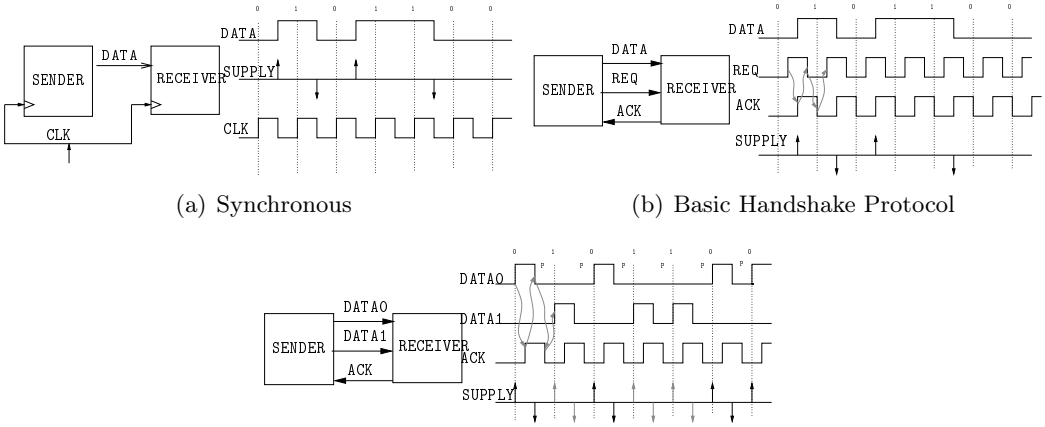


Figure 6: Flot de conception simplifié pour les FPGAs



(a) Synchronous

(b) Basic Handshake Protocol

(c) 1-out-of-2 4-phase

Figure 7: Les Protocoles Asynchrones

composants périphériques dans le SoC, sauf qu'il est reconfigurable. Pour faciliter cette reconfiguration, nous avons un autre composant FASE_CONF qui est en charge du contrôle de la configuration. Cette fonction est nécessaire pour la configuration à chaud de FASE.

Les figures 9(a) et 9(b) expliquent l'architecture logique et l'architecture de routage de FASE. L'architecture logique est similaire à un FPGA traditionnel sauf sur le point de la configuration nommé `RESET_MASK`. Ce point de configuration est utilisé pour distinguer la partie du FPGA qui est active de la partie qui en train d'être reconfigurée. Grâce à ce bit de configuration les parties actives ne sont pas initialisées chaque fois qu'un nouveau bloc est reconfiguré dans FASE. La figure 9(b) montre deux blocs dans le FPGA FASE, la partie rouge montre les éléments actifs alors que la partie bleue montre la partie sous reconfiguration. Les éléments logiques verts constituent l'interface entre ces deux blocs.

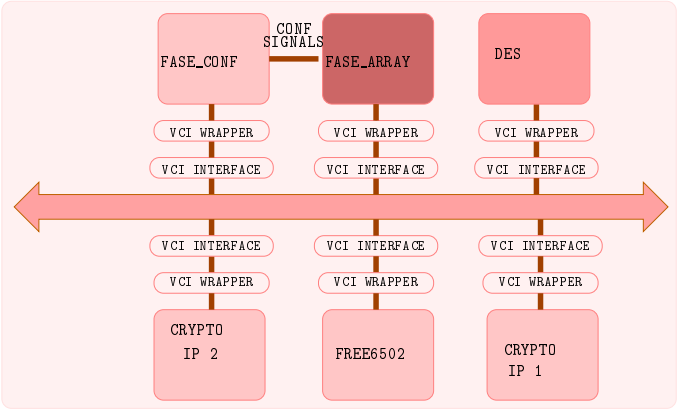
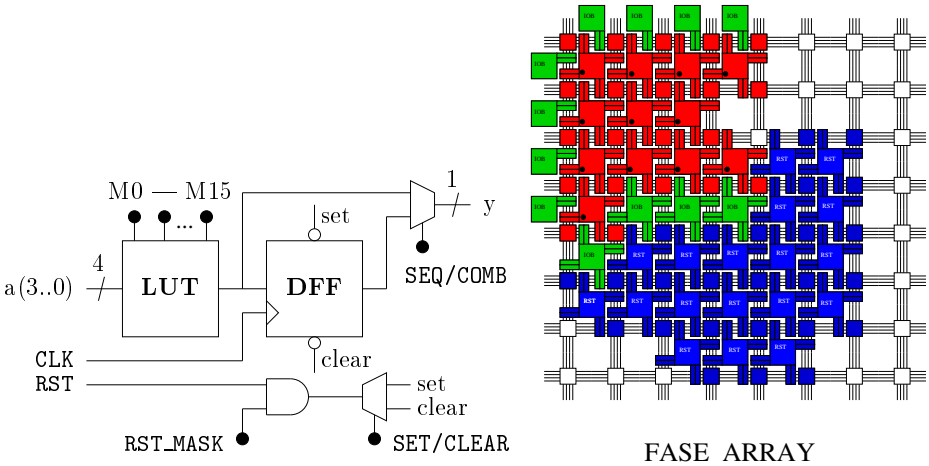


Figure 8: L'Interface VCI pour `FASE_ARRAY` et `FASE_CONF`.



(a) `CLB`: 4 → 1 `LE` avec masquage de reset.

(b) Deux Circuits

Figure 9: Reconfiguration "Run-Time" (RTR) de IP2 alors que IP1 est exécuté.

Flot de Conception

Nous utilisons les outils libres pour implémenter les circuits logiques dans FASE, comme expliqué dans la figure . D'abord les descriptions des circuits au niveau RTL sont synthétisés avec l'outil "Quartus" d'ALTERA. Ensuite les fichiers "VQM" générés par "Quartus" (qui contiennent les descriptions structurelles) sont transformés en format "NET" lus par l'outil de placement/routage VPR. VPR est un outil source ouvert développé à l'Université de Toronto. VQMTONET est un outil basé sur lex/yacc développé dans le cadre de cette thèse.

Prototype

La figure. présente le prototype de notre circuit SoC SECMAT (fabriqué en technologie 130nm de STMicroelectronics) contenant le FPGA FASE. Le prototype est un FPGA de dimension 8×8, avec 64 éléments logiques. Nous pouvons voir le FPGA en haut à gauche du circuit. Le contrôleur de configuration `FASE_CONF` se trouve dans la partie commune. La table 4 présente les données principales liées à ce prototype.

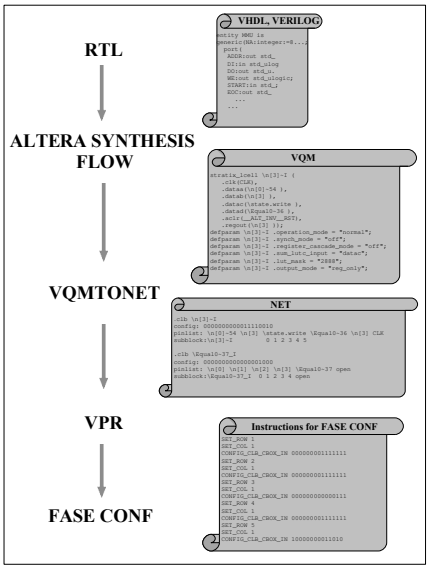


Figure 10: La Flot de Conception utilisé pour FASE

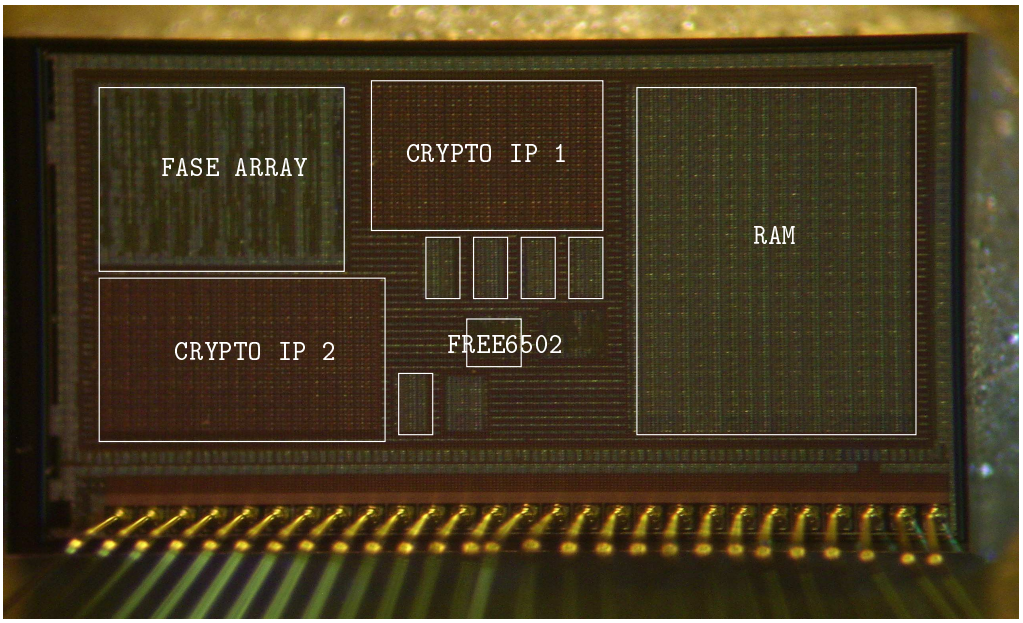
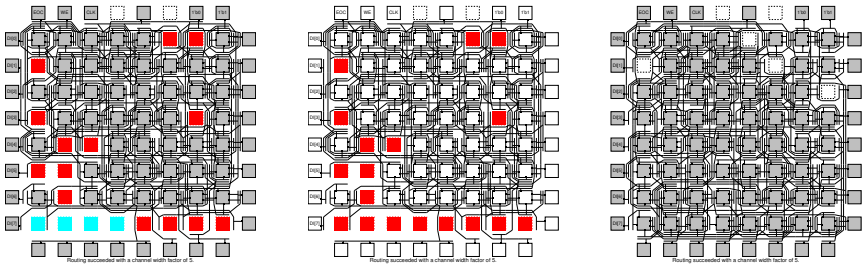


Figure 11: La micrograph de FASE‘

Table 4: Physical Details.

Die Size(SoC)	$4.1mm^2$
Area(FASE ARRAY)	$0.6mm^2$
Transistor Count	200,000 (Approx.)
V_{dd}	1.2V
V_{dd} IO	1.2V
F_{max}	32MHz



(a) IP2 in execution and IP1 CLBs & switchboxes are configured (red).

(b) IP2 erased.

(c) IP1 functional.

Figure 12: Reconfiguration "Run-Time" (RTR) de IP2 alors que IP1 est exécuté.

Expériences

Le FPGA FASE est conçu pour implémenter des contre-mesures basées sur la reconfiguration dynamique modifiant le comportement du calcul et ainsi permettant de contrer les attaques par canaux cachés. On montre la faisabilité des ces contre-mesures avec l'expérience décrite dans ce chapitre. D'abord nous utilisons deux blocs synthétisés par le flot de conception spécifique à FASE :

- P1: Exécute un cipher de Vernam avec les 256 octets lus à partir de la RAM et écrit le résultat dans la RAM. La clé est codée dans l'IP.
- IP2: Un compteur 24 bits. Il stocke la valeur dans la RAM

La séquence utilisée pour vérifier la reconfiguration à chaud est la suivante :

- D'abord on configure l'IP2 dans FASE (voir fig. 12(a))
- Pendant que l'IP2 s'exécute on configure l'IP1 dans les éléments logiques qui ne sont pas utilisées par l'IP2
- Une fois que l'exécution de l'IP2 est terminée, on configure le reste de l'IP1, et on exécute l'IP1.

- On vérifie que les résultats sont corrects.

Les résultats de cette expérience sont décrits dans la table 5.

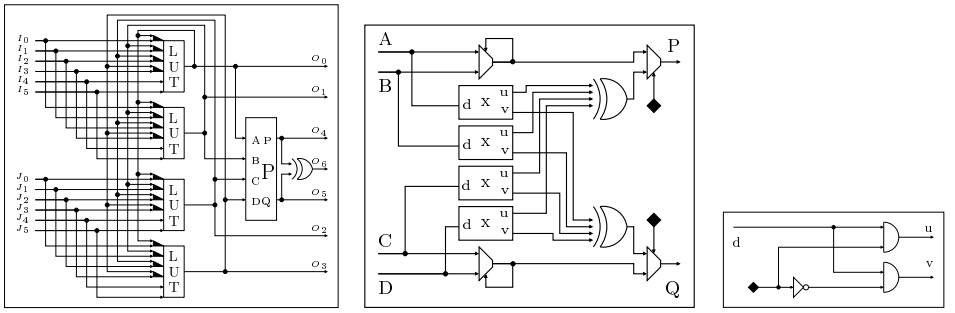
Table 5: Démonstration of RTR on 8x8 RGA.

Hardware Blocks	**No of CLBs**	**Execution Time**	**Reconfiguration Time**	**Erase Time**	**Reconfig Ratio**
	LUT4+FF	(Computation Cycles)	(Computation Cycles)	(Computation Cycles)	$\frac{\text{Column III}}{\text{Column IV}}$
IP1	58	1024	3712	174	0.276
IP2	44	67108864	2816	132	23831.27

SAFE: Architecture d'un FPGA Asynchrone

Le chapitre 7 de ce manuscrit de thèse présente l'architecture d'un FPGA asynchrone. Nous avons déjà indiqué les avantages d'utiliser un FPGA asynchrone pour les contre-mesures.

Architecture

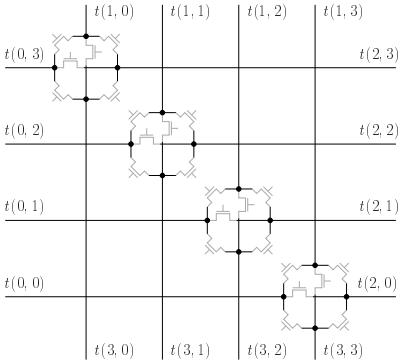


(a) PLB Functional Overview (b) Block P in Detail (c) Block X in Detail

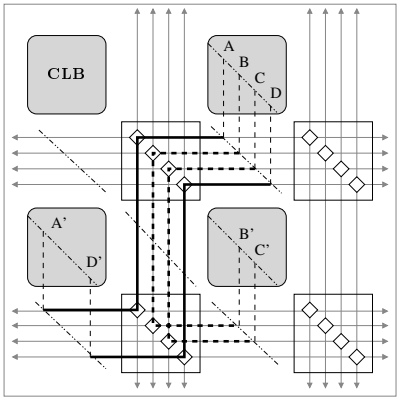
Figure 13: Vue d'ensemble du PLB

Architecture Logique La figure 13 présente l'architecture logique du FPGA asynchrone. Le PLB (Programmable Logic Block) est conçu pour implémenter plusieurs types de codage et de protocoles de la logique asynchrone. La table ci-dessous décrit le nombre de PLB utilisés pour implémenter chacun d'entre eux.

	A	B	C
2-phase EDGE	1	n.a.	n.a.
2-phase LEDR	0.5	1	n.a.
4-phase	0.5	1	2



(a) Subset switchbox using six-way switchpoints.



(b) Equitemporal lines for subset switchbox routing.

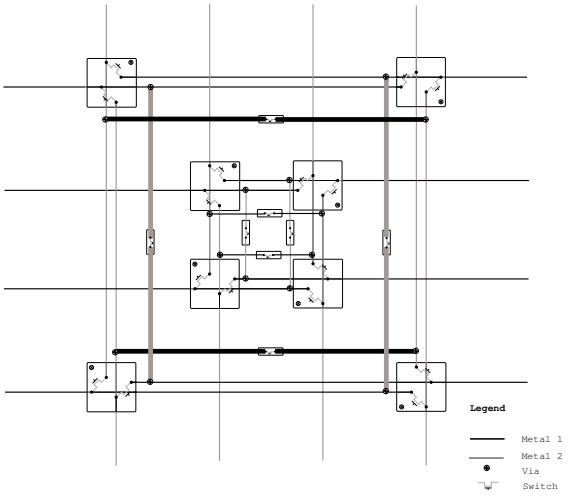
Figure 14: Boîte de connexion (switchbox) de type subset.

Architecture de Routage La logique asynchrone est une contre-mesure statique contre les canaux cachés. Pour cette raison, et aussi parce que chaque connexion dans un FPGA est chargée de beaucoup de switches, il faut équilibrer les connexions entre les équipotentielles dual-rail. Notre architecture de routage pour le FPGA SAFE est conçue à cet effet.

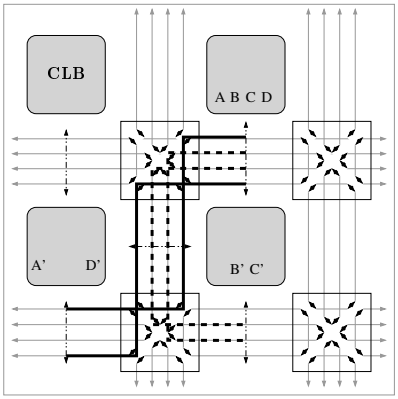
La figure 14(a) présente les enjeux de l'équilibrage dans une architecture de routage avec les switchbox (matrices de commutation) de type "subset". Les signaux sur les lignes équitemporelles ont le même retard, c'est à dire que le signal sur une ligne équitemporelle arrivera au même moment que le signal sur une autre ligne équitemporelle. Dans la figure 14(b) nous voyons que dans cette architecture les lignes équitemporelles sont en diagonale. Pour cette raison les connexions sur les canaux de routage sont également implémentées en diagonale. Dans la figure 15(b) on retrouve une boîte de commutation de type "paire torsadée". L'avantage de cette boîte de commutation est de router les fils en paire torsadée pour procurer une bonne immunité contre les attaques qui observent le rayonnement électromagnétique du circuit cryptographique. Il faut aussi équilibrer les connexions établies au travers des boîtes de connexions (switchbox), ce qui dépend du type de boîte de commutation utilisée dans le FPGA. Il est évident qu'à partir des lignes équitemporelles dans les figures 15(b) et 14(b), les switchs doivent être placés sur ces lignes équitemporelles. En conséquence quand on utilise les boîtes de commutation de types subset, les switchs sont placés sur une diagonale et pour les boîtes de commutations de type paire torsadée sur une ligne perpendiculaire.

La figure 16(a) présente le layout CMOS d'une boîte de commutation pour le type subset. Nous pouvons constater que les distances entre les entrées et les switches sont toutes les mêmes, idem pour les distances entre les switches et les canaux de routage.

Architecture de Configuration La figure 17 présente l'architecture de configuration de notre FPGA Asynchrone. Cette chaîne est constituée d'une série de cellules "Full-Buffer", similaire à un registre à décalage dans le domaine synchrone. Ces cellules "Full Buffer" communiquent entre elles avec un protocole de types poignée de main (handshake) dites 1-out-of-4. Ce protocole est rapide et fiable car il n'y a pas de contraintes liées à une horloge centrale.

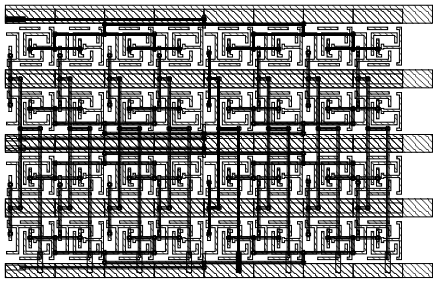


(a) Twisted Pair switch matrix layout scheme



(b) Equitemporal lines for the twisted-pair switchbox.

Figure 15: La boîte de commutation de type "paire torsadée".



(a) Sample cbox layout.

Figure 16: Layout Semi-Custom pour Boîte de Connexion.

Prototype

La figure 18(a) montre le circuit 3×3 prototype de notre FPGA asynchrone. Cette puce est fabriquée en utilisant le processus ST Microelectronics 65 nm 7-couche appelée CMOS065. La largeur du canal (nombre de canaux) du FPGA est 8, et il y a 9 entrées/Sorties pour chaque côté. La figure 18(b) montre comment les segments sont acheminés à l'intérieur du FPGA. Ce FPGA a été conçu à l'aide d'un flux automatique tel que décrit dans [137]. Par rapport aux contraintes d'équilibrage et de connexion, nous avons utilisé la méthode suivante:

- Nous avons d'abord placé les interrupteurs, de manière symétrique, comme indiqué dans les schémas auparavant.
- Tous les segments de canaux sont routés manuellement pour atteindre l'équilibre nécessaire.
- Les points mémoire de configuration et les signaux sont placés/routés automatiquement.

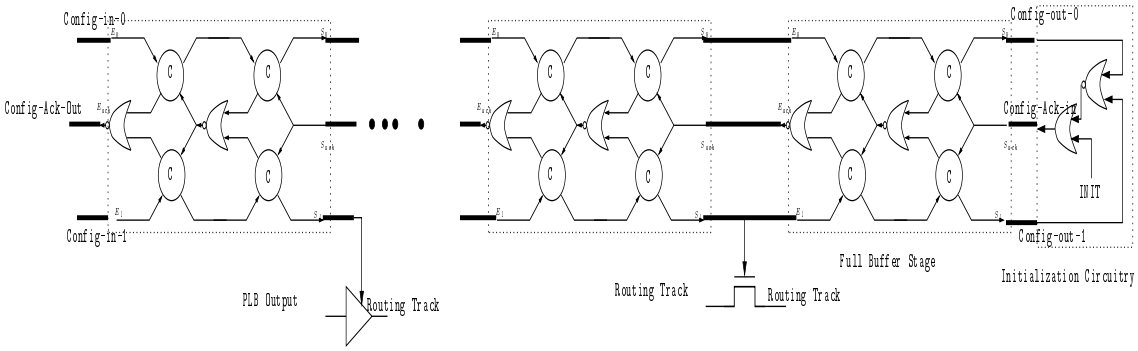


Figure 17: La Chaîne de Configuration Asynchrone avec circuit pour Reset

Du fait que la taille du FPGA dépend principalement de la taille des commutateurs et des points mémoire de configuration, l'incorporation du routage équilibré ne se traduit pas par une surcharge en termes de superficie. Le prototype occupe une superficie de $1111.6000\mu m \times 947.6000\mu$ m dans le silicium et contient environ 200.000 transistors.

Résultats Expérimentaux

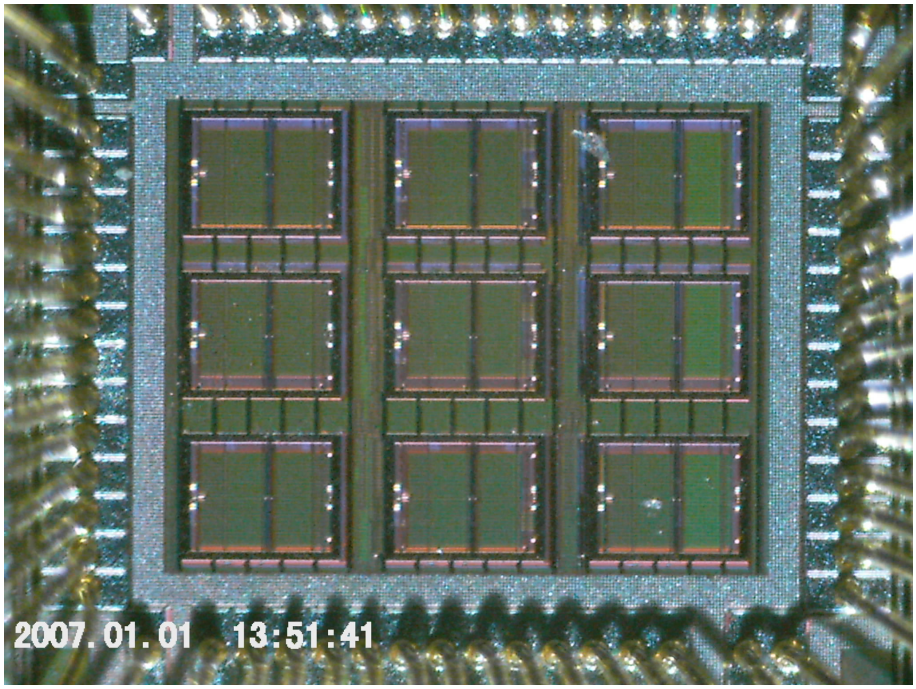
Configuration La séquence de cette expérience est la suivante:

- D'abord la chaîne de configuration de sécurité est initialisé, en mettant le signal *INIT* à l'entrée du SAFE à '0', puis libéré après un certain temps. D'abord la chaîne de configuration de sécurité est initialisée, en mettant le signal *INIT* à l'entrée du SAFE à '0', puis libéré après un certain temps.
- Le débit binaire de fichier généré par VPR est chargé dans une mémoire RAM intégrée à la plate-forme de test DE2, à partir du PC. Ce flux binaire est alors converti en codage asynchrone 1-out-of-2 et envoyé aux signaux *configuration-0* et *configuration-1*de SAFE.
- Le Contrôleur de la carte DE2 surveille la sortie *acknowledge* de SAFE, et met une nouvelle valeur dans la chaîne de configuration suivant le protocole 4-phase. Il compte également le nombre d'accusés de réception. Pour une configuration réussie, il doit recevoir exactement $4692(HEX1254)$ accusés de réception.

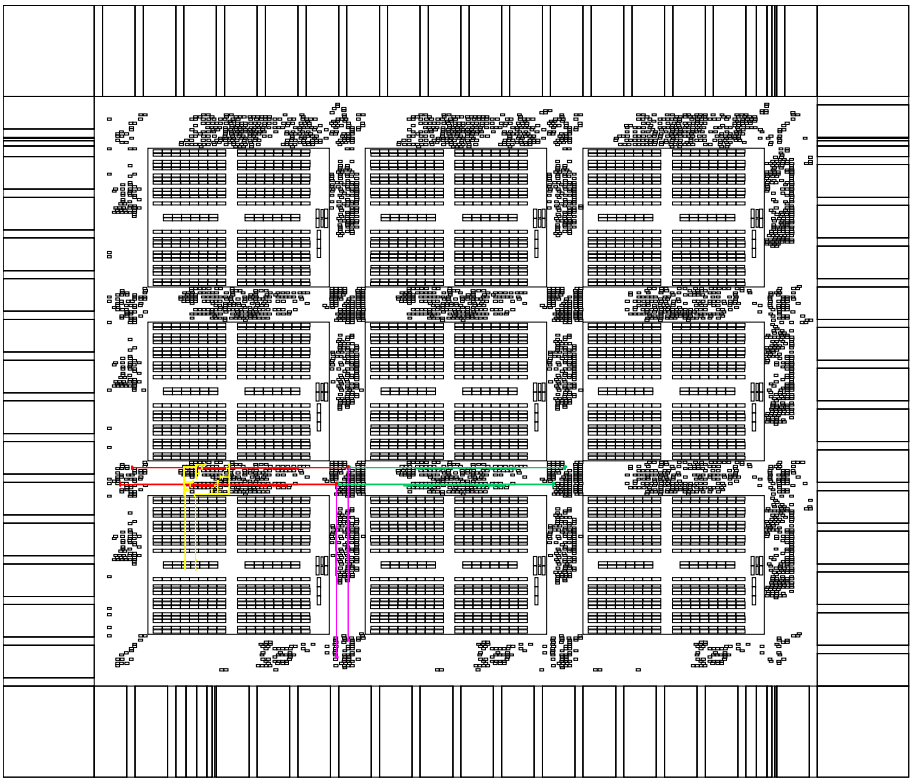
Les résultats de cette expérience sont les suivantes : Lorsque le train binaire ne contient que des '0's la configuration est réussie avec une vitesse très élevée (fréquence jusqu'à $50MHz$). Quand le débit binaire contient des '1's la configuration réussit mais à une vitesse faible (autour de $10KHz$) Malgré le fait qu'en raison du codage asynchrone les '0's et '1's sont équivalents en termes de transition, nous pensons que ce comportement peut être du à un bug que nous allons essayer d'expliquer avec des simulations au paragraphe 7.9.4.

"Hop Mismatch"

Le but de cette expérience est de d'estimer le déséquilibre entre les deux rails constituant le double rail des variables asynchrones. Ceci est très important en termes de robustesse, car un "side-channel attack" peut exploiter cette différence. La contrainte de consommation

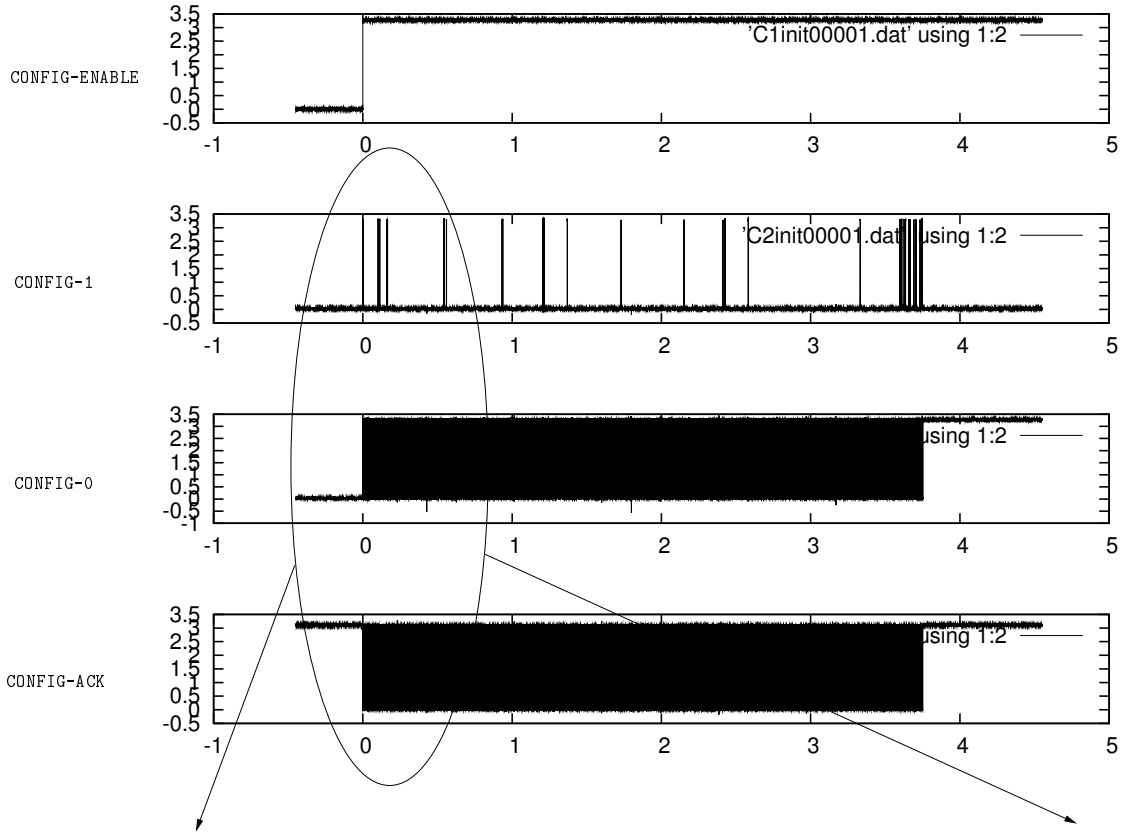


(a) 3x3 Prototype Asynchronous FPGA (CMP Run S65C8_1)

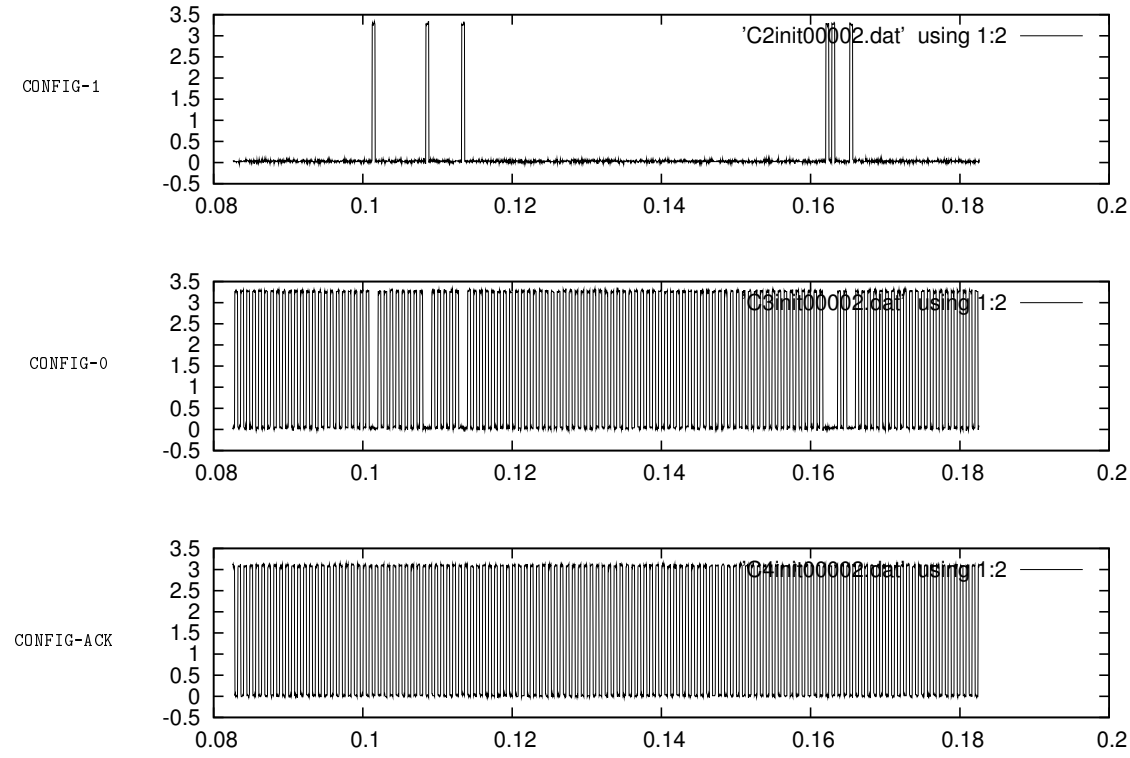


(b) Routes inside the FPGA

Figure 18: Prototype

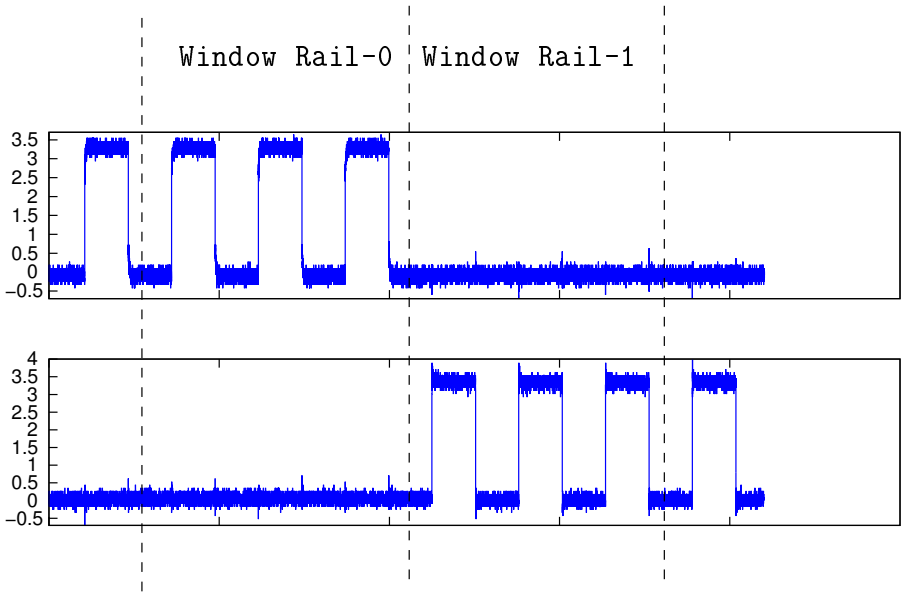


(a) The Whole Configuration Trace 4692 bits

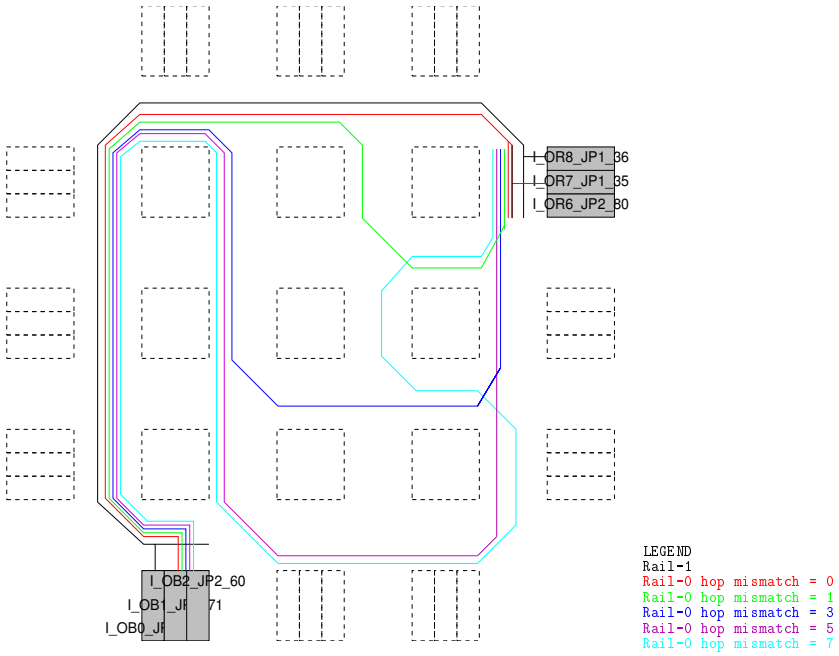


(b) Zoomed

Figure 19: Measured Trace of Configuration signals



(a) Measuring the Difference between rail-0 and rail-1 for different lengths of rail-0



(b) After Processing

Figure 20: Description de l'expérience "Hop Mismatch"

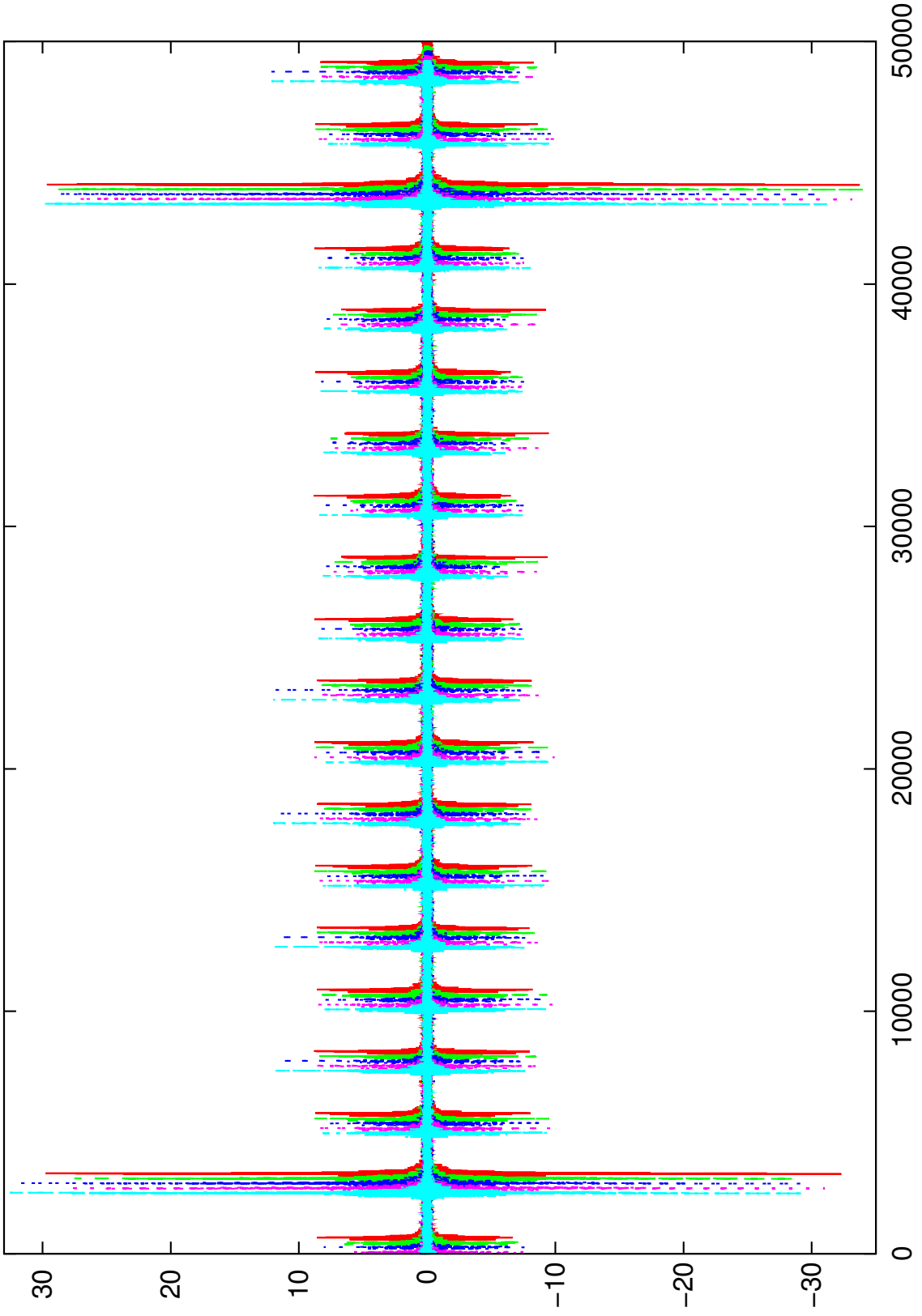


Figure 21: The results for different values of Hop Mismatch(Superposed and staggered)

constante dépend de cet équilibre. Les figures 20(a) et 20(b) décrivent le dispositif expérimental. Nous avons un rail de deux voies dans le FPGA. Le temps de parcours du RAIL-1 de ce double rail, reste constant durant l'expérience. Le temps de parcours du RAIL-0 est modifié par le nombre de "HOP" (commutateurs) compris dans cet ensemble (0,1,3,5,7) par rapport au RAIL-1, comme le montre la figure 20(b) . Nous envoyons 4 impulsions pour RAIL-1 et RAIL-0, et on mesure la consommation électrique. Cette mesure se fait avec un signal de déclenchement séparé. W1 (t) est la trace de RAIL-1 et W0 (t) est la trace pour RAIL-0 dans cette fenêtre. Le déséquilibre entre ces deux traces est ensuite calculé avec :

$$Balance = \frac{RMS(W_0(t)}{RMS(W_1(t)}$$

Les traces correspondantes sont présentées dans 7.25. Nous pouvons voir les instants de déclenchement de la mesure. Les traces d'une éventuelle dissymétrie suivant divers "hop" et les valeurs de déséquilibre sont indiquées dans 7.25. La figure 7.26 montre les mêmes traces superposées en quinquonce de façon à les comparer. Ils utilisent le même code de couleur que dans la fig. 7.24(b).

Hop Mismatch	Balance
0	1.106
1	1.120
3	1.158
5	1.119
7	1.173

Conclusion

Ce manuscrit de thèse est à l'intersection de trois sous-domaines différents dans le domaine de l'architecture informatique moderne: Cryptographie, Architectures FPGA et Circuits Asynchrones, avec l'objectif spécifique de trouver une contre-mesure adaptée face aux attaques physiques sur du matériel cryptographique. Les contre-mesures statiques sont incorporées durant le temps de conception du circuit, et s'appuient sur une puissance constante de signalisation, c'est à dire la consommation d'énergie de façon à être indépendante du calcul sous-jacent. D'autre part, les contremesures dynamiques permettent de modifier aléatoirement le comportement du calcul mais nécessitent une architecture de reconfiguration particulière dite "à chaud". Cette thèse tire parti des deux approches, en combinant leurs avantages, et essaie de gérer efficacement les inconvénients. Pour atteindre cet objectif, nous avons fabriqué deux prototypes de FPGA :

- Un FPGA embarqué (eFPGA) dans la technologie 130nm qui a été testé fonctionnellement et a permis de démontrer l'intérêt de reconfiguration partielle à chaud,(*RTR*). Ce eFPGA reconfigurable peut être utilisé à l'avenir pour évaluer la faisabilité des contre-mesures en utilisant une méthode *RTR*. Un FPGA embarqué (eFPGA) dans la technologie 130nm qui a été testé fonctionnellement et a permis de démontrer l'intérêt de reconfiguration partielle à chaud,(*RTR*). Cet eFPGA reconfigurable peut être utilisé à l'avenir pour évaluer la faisabilité des contre-mesures en utilisant une méthode *RTR*.

- Un FPGA totalement asynchrone en technologie 65nm, en collaboration avec *TIMA, Grenoble*, permettant d'évaluer les contre-mesures à la fois dynamiques et asynchrones, à l'aide de plusieurs protocoles asynchrones. Ce FPGA stand-alone a été testé au niveau de la fonctionnalité de la chaîne de configuration et des interconnexions. Les résultats de simulation sont en conformité avec les résultats des tests pour les interconnexions.

En conclusion nous pouvons mettre à profit les contre-mesures étudiées dans cette thèse, qu'elles soient dynamiques et statiques pour augmenter la robustesse des fonctions cryptographique face aux attaques par canaux cachés. Les solutions étudiées donnent des moyens pour augmenter le MTD (ou Nombres de Mesures pour obtenir la clé secrète), ainsi que la résistance face aux attaques en fautes.

Perspectives

Comme nous l'avons vu dans les sections précédentes de ce manuscrit de thèse, les contre-mesures sont souvent classées selon leur MTD (Nombre de Mesures pour obtenir la divulgation du secret). Idéalement, une contre-mesure devrait avoir un MTD si haut qu'il n'est pas possible pour un attaquant d'obtenir la clé secrète dans la limite raisonnable de temps et d'espace. Une direction de recherche possible est d'être constamment à la recherche des contre-mesures à haut MTD, ce qui a motivé le principe de cette thèse. Le MTD dépend non seulement de la contre-mesure mais aussi de l'avancement des équipements et de l'efficacité des techniques de traitement. Une autre direction de recherche est de limiter le nombre d'utilisations d'un dispositif cryptographique à l'avance. Dans tous les attaques connues, l'attaquant doit utiliser l'appareil un certain nombre de fois pour deviner la clé, ou construire un modèle de l'appareil concerné. Les défis principaux de recherche dans cette approche consiste à prédire avec précision la durée de vie de périphérique, donc une compréhension accrue du processus de dégradation. Nous pouvons imaginer l'utilisation de compteurs numériques dans le circuit, mais les attaquants peuvent essayer de pirater ces compteurs. Pour atténuer cette attaque potentielle, il est proposé d'étudier des solutions limitant la durée de vie des dispositifs cryptographiques, à partir de mécanismes de dégradation intrinsèque [131]. Nous présentons brièvement dans cette section en tant que sujet de recherche futur, certains des processus physiques possibles que nous pouvons exploiter pour parvenir à limiter la durée de vie des dispositifs, comme par exemple l'ElectroMigration, le NBTI et le Memristor.

Introduction

Cryptography is a mean to defend against potential attackers, notably to protect confidentiality, integrity or secure authentification, whereas cryptanalysis is about the challenge to retrieve hidden information. The age-old Arms Race between these two domains continually takes new dimensions with the ever changing technology, on the top of which these algorithms are implemented. There are no known mathematical cryptanalysis methods which can decrypt standard cryptographic algorithms like AES faster than the exhaustive Brute-Force attack assuming that the cryptanalyst has access to both plain-text and encrypted messages. However all such algorithms are implemented with some physical process, and access to information leaked from these physical processes, makes the job of the cryptanalyst much easier. These kinds of information leakage from physical processes are commonly known as Side-Channel Leakage.

For the purpose of this manuscript, we divide cryptanalysis broadly into two categories: mathematical and physical. In this manuscript, we assume that concerned cryptographic algorithms are secure at the mathematical level, and we will specifically address the issue of physical cryptanalysis and counter-measures. Physical cryptanalysis can again be of two types, namely active and passive. Injecting faults to perturb the physical implementation is an example of active attacks, whereas attacks based on measuring power consumption/EM radiation are examples of passive attacks, commonly known as Side-Channel Attacks.

Physical cryptanalysis has been demonstrated to be effective against various standard algorithms, and on various platforms in recent times. Researchers have shown that side-channel attacks can be mounted on standard cryptographic algorithms like DES [21], AES [32], RSA [21]. References [50, 44] provide the details of such attacks on FPGA implementations whereas various attacks [47] has been reported on ASIC implementation. A widely known Side-Channel Attack is DPA (Differential Power Analysis) [22], which exists in various forms [37] and concerns the information leaked through supply current peaks. Attack which exploit the Electromagnetic Emissions (EMA) [11, 1] from the hardware, constitutes another major branch of Side-Channel Attacks. The Timing Attacks [21] which uses the difference in execution time, as its major source of information has also been reported. The reader could as well find a comprehensive report of active attack details in [5].

Now then, who's at risk? A very evident answer should be banking applications. Credit cards use algorithms similar to RSA for authentication, and 2-key Triple DES for the challenge [49]. Wholesale frauds on systems which rely on smart cards for their security(e.g. Pay-TV) could well be a target of such attacks. Mounting a side-channel attack calls for considerable expertise. So such techniques are prone to be used when there is a considerable gain. A major threat could be intellectual property protection because of this reason. The attacker can easily gain access to piracy protection devices embedded in commercial systems, and steal the IPs. All the more reason to incorporate side channel

resistance into these systems.

The rest of this manuscript is organized in three parts. In the first part I we briefly explain the different domains of research, related to this manuscript. Chapter 1 provides an introduction to cryptanalysis, with emphasis on physical cryptanalysis. We also discuss an analytical model of the side-channel. Chapter 2 acquaints the reader with modern FPGA architectures and associated CAD tools, we also touch upon the subject of rout ability analysis of netlists in a gate array. The essential know how of asynchronous circuit design is presented in chapter 3, and in chapter 4 we present the recent research efforts in the concerned domains, and our motivations.

Part II documents the engineering associated with this research effort. In chapter 5 we present the overall design methodology used for realization of FPGAs in CMOS. Chapter 6 presents a embedded run-time reconfigurable FPGA architecture and experiments for dynamic countermeasures. Chapter 7 describes the design of an asynchronous FPGA intended for static countermeasures.

Part III presents the conclusion and provides the reader with some food for thought, as how to limit the life of a cryptographic device in order to prevent the attacker from gaining excessive information about its implementation.

Part I

Background Art

Summary

This part of the manuscript states the existing research, and the principles to acquaint the reader to the problem. The first chapter 1 explains the intricacies of side-channel attacks and elaborates various methods. The second chapter 2 explains the logic and routing architecture of island-style FPGAs, as well as the CAD algorithms associated with it. This chapter also includes an introduction to routability analysis in FPGAs based on Rent's Rule. The third chapter 3 explains the basic principles of asynchronous circuit design, and provides some examples of basic cells. It also outlines related synthesis methods. In the last chapter 4 we present recently published results and architectures in the above-mentioned domains for the sake of comparison with this thesis.

Chapter 1

Physical Cryptanalysis

1.1 Cryptography

1.1.1 Kerckhoff's Principle

"A cryptosystem should be secure even if everything about the system, except the key, is in public knowledge"

This is one of the guiding principles, for the designer of a cryptosystem as opposed to *"Security through Obscurity"*. This is also stated in another form "*the enemy knows the system"* known as Shanon's maxim.

We state this principle at the beginning of this thesis manuscript, since hereafter we will assume implicitly, that everything about the cryptosystem is known, if not stated otherwise.

1.1.2 Classification of Ciphers

Modern day Cryptography consists of two very broad classes: namely *Secret Key Cryptography* and *Public Key Cryptography* Secret Key Cryptography is the conventional cryptographic system where the secret key is known to the encrypter and the receiver and nobody else. In Public key Cryptography, the sender encrypts the data with receiver's public key. The receiver on the other end, decrypts this data with his/her private key.

1.1.2.1 Secret Key Cryptography

Secret key cryptography consists of two major classes, namely block and stream cipher.

Block Cipher Figure 1.2(a) depicts the generic block cipher system. The serial plain text data is converted to blocks of fixed size, which are then encrypted with the same key. Finally the encrypted blocks are converted to serial form. Very well known standards are block ciphers such as DES [30], AES [31]. We can see from figure 1.2(a) that if two identical blocks are encrypted, the results will be identical. This leaks some information about the identicalness of the blocks. To circumvent these problems, often block ciphers accommodate an additional mode of operation where result of i_{th} encryption is used as the key for $(i+1)_{th}$. This is also known as CBC (Cipher Block Chaining) mode.

Examples of block ciphers are present everywhere in our day-to-day life. The UNIX "crypt" program encrypts 64 '0' s with 25 rounds of DES, where ouput of each round is used as the key for the next round. with the first key derived from user password (e.g.

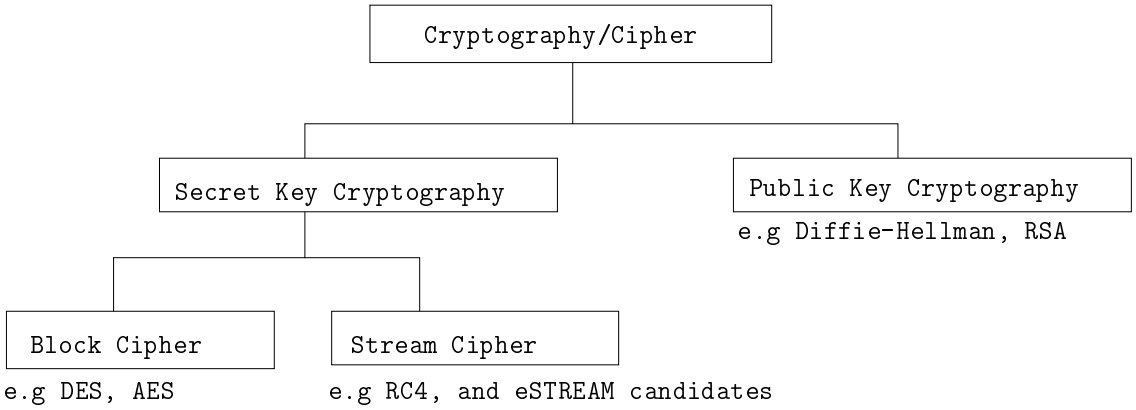
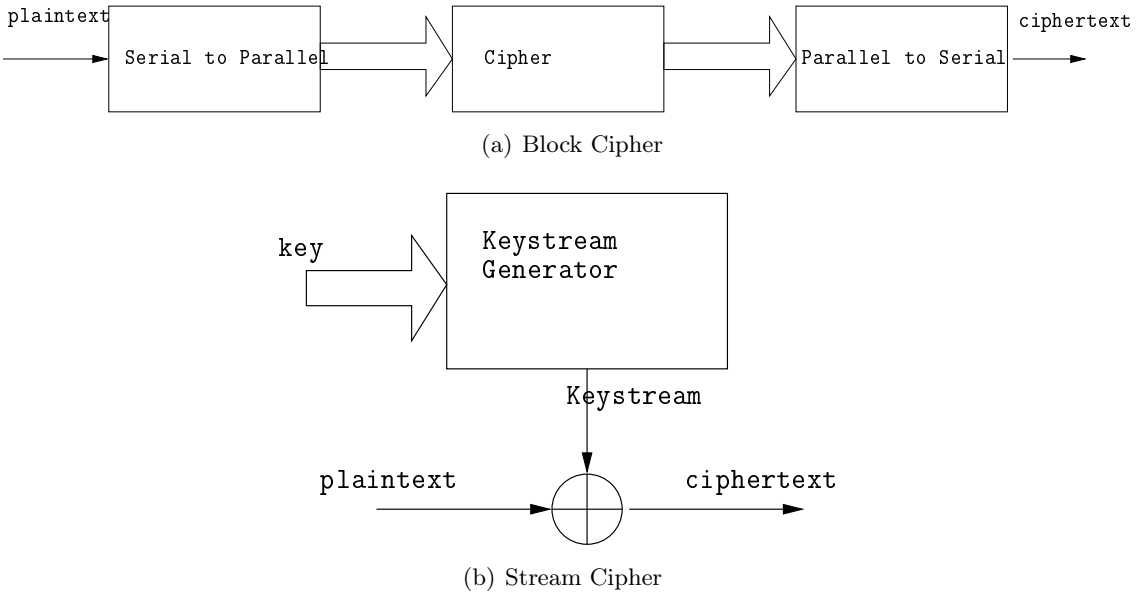


Figure 1.1: Classification of Ciphers.



(a) Block Cipher

(b) Stream Cipher

Figure 1.2: Block and Stream Ciphers

The first eight characters of the user password) [29]. EMV (EuroPay Master Visa) credit cards use "Triple DES" to encrypt communications with the bank.

Stream Cipher A stream cipher applies a time-varying transformation to the individual bits of the plain text. Figure 1.2(b) depicts the most popular form of stream ciphers, where the cipher bits are generated by simple modulo-2 addition of XORing the `plain text` and `key stream` bits. This `key stream` is generated by a `key stream Generator` which is controlled by the `key`. Generally the key stream length is much longer than the key. An international contest to select string stream ciphers in on-going in Europe: http://www.ecrypt.eu.org/stream/.

1.1.2.2 Public Key Cryptography

A Public Key Cryptographic system is described by two sets of algorithms that compute invertible functions. Let these two sets of algorithms be denoted as *public* and *private*. The invertible transformations computed by these algorithms are:

$$\begin{aligned} \textit{public}{:}f(x) &= y\,, \\ \textit{private}{:}f^{-1}(y) &= x\,, \end{aligned}$$

where x is a certain message, and y is the corresponding cryptogram. The fundamental requirement of a public key cryptography system is that the function f must be a *trapdoor one-way function*. The term one-way refers to the fact that for any message x it is easy to compute $f(x)$ from the knowledge of the algorithm "*public*", but given a cryptogram y it is extremely difficult to compute the inverse $f^{-1}(y)$. On the other hand, an authorized user with the knowledge of the algorithm "*private*" would find it very easy to compute the inverse $f^{-1}(y)$. Thus the "*Private*" algorithm provides a trapdoor for the authorized user. In a Public Cryptographic System, the "*public*" algorithm is known to everybody, but only the authorized user possess trapdoors or "*Private*" algorithm.

Diffie-Hellman Diffie-Hellman Secret Key Exchange System uses the fact that it is easy to calculate a discrete exponential, but difficult to calculate a discrete logarithm. For more details see [16].

RSA Most successful implementation of public-key cryptography is the Rivest-Shamir-Adleman system, named after its inventors. The RSA algorithm is a block cipher based on the fact that finding a random prime number of large size (100 digit) is computationally easy, whereas factoring the product of two such numbers is currently considered to be computationally infeasible. It is considered to be one of the most secure cryptographic algorithms, since it withstood several mathematical cryptanalysis attacks. However this does not make it secure against Side-Channel Attacks, since these attacks are based on physical information leakage.

1.2 Cryptanalysis

In principle any method (Mathematical or otherwise) that can reduce the number of encryptions needed to retrieve a hidden key, compared to the exhaustive key-search is considered as a cryptanalytic method. At a higher level cryptanalysis or attacks can be classified as "ciphertext only", "known plaintext", and "chosen plaintext". For example, in ciphertext only attack, the attacker retrieves the key just by analyzing the output of the cryptographic system. Figure 1.3 compares these methods w.r.t the mathematical difficulty and the logistics needed. It's obvious that ciphertext only attacks can be mounted with less effort(e.g by eavesdropping on a communication channel) whereas chosen plaintext attacks require that the cryptanalyst gain the control of the cryptographic device.

1.2.1 Classification of Cryptanalysis

For the purpose of this thesis manuscript, we have divided cryptanalysis into two categories, namely mathematical and physical. We will mostly focus on physical cryptanalysis

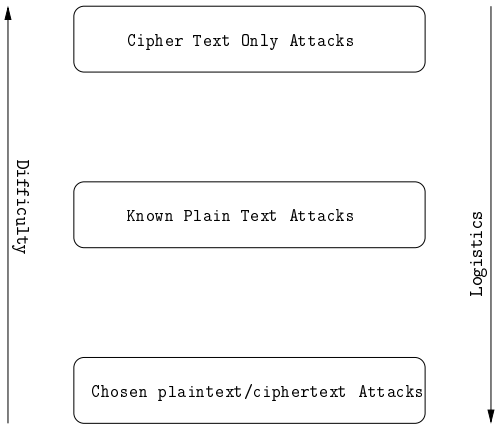


Figure 1.3: Difficulty and Logistics.

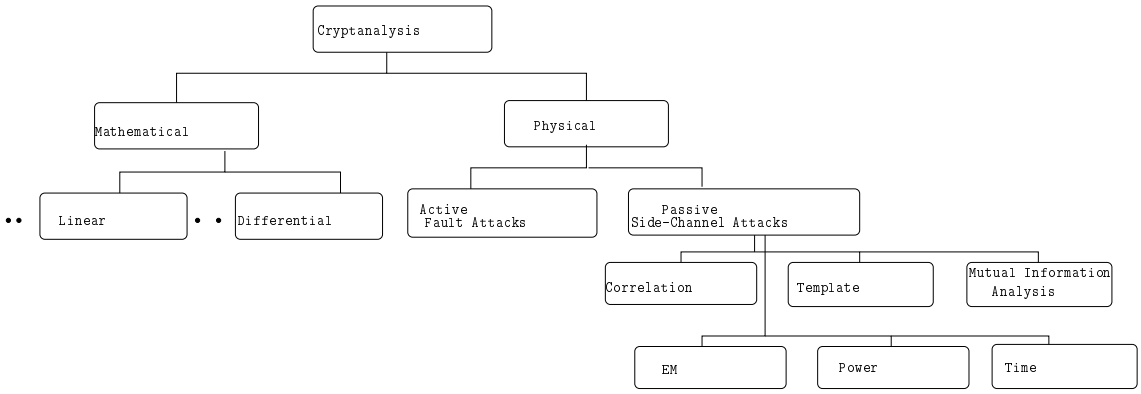


Figure 1.4: Classification of Cryptanalysis.

techniques and in particular to Side-Channel Attacks. We briefly describe the mathematical cryptanalysis techniques in this section. The reader may note that this classification is completely artificial, and the power of mathematical cryptanalysis can be incorporated into side-channel attacks, to render them more efficient.

1.2.1.1 Mathematical Cryptanalysis

Differential Cryptanalysis Differential Cryptanalysis is a powerful mathematical cryptanalysis technique applicable to a wide variety of Cryptographic algorithms. Also known as T-Attack to IBM design team which designed DES, the successor of Lucifer, one of the major criteria in designing DES was to thwart this type of attacks [9].

A differential cryptanalyst trying to break the system is in possession of large amounts of plaintext and corresponding ciphertext. He/She knows the complete specification of the system (IP, S-boxes, P, key schedule). He/She starts with two messages, m and m', differing by a known difference δm.

$$\delta m = m \oplus m_i$$

Now suppose that there is a relation between input differences and output differences for some S-box. That is, the 64 possible 6-bit inputs to S, can be divided into 32 pairs, so that the XOR of the two inputs in each pair is the constant value δI_i because this is the change

of inputs on the ith S-box. For each such pair of inputs, consider the pair of outputs and consider their XOR called δO_i.

Differential cryptanalysis depends on the fact that many input pairs with a given input difference δI_i,j give rise to the same output difference δO_i.

For example, for the first S-box(which has 6 input bits and 4 output bits) in DES with δI_i = "110100", only eight of the 16 possible values of δO_i can occur, and one value of δO_i = ""0010" occurs for eight of the 32 input pairs sharing the difference δI_i = "110100". (see [7] for more details).

Differential cryptanalysis was first published in the year 1990 by Biham and Shamir, with cryptanalysis of FEAL and 16-round DES cipher.

Linear Cryptanalysis Linear Cryptanalysis, first published by Mitsuru Matsui [26], is a more recent and more powerful attack. It requires known plaintexts only, rather than chosen plaintext as in the case of Differential cryptanalysis (Thus one step closer to the ciphertext only attack), and can break 16-round DES in less texts(2^{47}). The number of texts is even less for "ASCII only" or "English Sentences only" texts.

A linear cryptanalyst tries to find effective linear expressions for a given cryptographic algorithm which is of the form

$$P[i_1, i_2, \ldots i_a] \oplus C[j_1, j_2, \ldots j_b] = K[k_1, k_2, \ldots k_c]$$

where $i_1, i_2, \ldots i_a$,$j_1, j_2, \ldots j_b$,$k_1, k_2, \ldots k_c$ denote fixed bit locations, and the equation holds with a probability $p \neq \frac{1}{2}$, for randomly given plaintext P and the corresponding ciphertext C. K denotes the key value.

The effectiveness of the linear equation is defined as E ($E = |p - \frac{1}{2}|$). The basic algorithm is as follows:

Step1:
Let T be the number of plaintexts such that the left side of the equation is zero.

Step2::
if $T \geq \frac{N}{2}$ (N denotes the number of plaintexts)
then guess $K[k_1, k_2, \ldots k_c] = 0$ (when $p > \frac{1}{2}$) 1 (when $p < \frac{1}{2}$),
else guess $K[k_1, k_2, \ldots k_c] = 1$ (when $p < \frac{1}{2}$) 0 (when $p > \frac{1}{2}$),

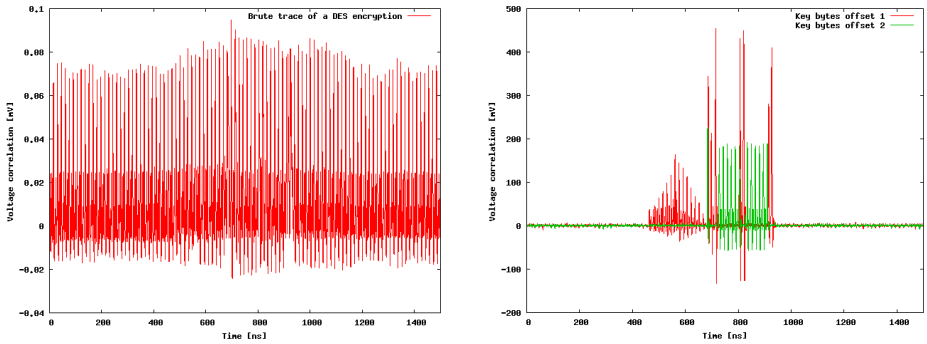
For more details see [26]. The main reason we mention these attacks in this manuscript, is the fact that effectiveness of Side-Channel Attacks can be enhancedby using these mathematical techniques.

1.2.1.2 Physical Cryptanalysis

We classify Side-Channel Attacks in different ways. Based on the acquisition method:

- Supply current measurement (DPA)
- EM emission measurement (EMA)
- Timing difference measurement(Timing Attack)

or based on processing methods:



(a) Raw Power Traces after Acquisition (b) After Processing

Figure 1.5: Peaks appear for the right guess

- Correlation based. (The measured traces are correlated with the predicted trace from assumed model)

- Template. (The model itself is created from experimental measurements on one sample for different key values, which is then used to compare the traces for clone circuits [37])

We will give a very basic example of how a side-channel attack is carried out, a broad overview can be found in [47]. Referring to fig 3.1(a) we can see that in an unprotected synchronous logic, each change of state of a signal can be distinguished by a current spike, and no change of state by an absence of current spike in the power supply line. Given this basic behaviour of CMOS circuits a side-channel cryptanalyst could proceed in the following fashion:

- He finds a signal which is a function of say N bits of the key, and the input message of M bits.

- He performs the encryption 2^M for each different message value and acquires the power trace of each of them.

- He makes 2^N key guesses and for each key guess he make current spike predictions for 2^M traces.

- Among these 2^N current spike predictions for 2^M encryptions, the one which correlates best with the measured power trace, is the correct key guess.

The activity of other nodes are not correlated since they are not a function of the targeted message and key bits. These activities of other nodes appear as noise after the processing of acquired traces. The resistance to physical cryptanalysis, relies on maximizing this signal to noise ratio(SNR) either by static or dynamic countermeasures. Figure 1.5(a) shows the raw power traces after acquisition on an ASIC implementation of DES, and figure 1.5(b) shows the appearance of predicted peak when correlated with the right key guess.

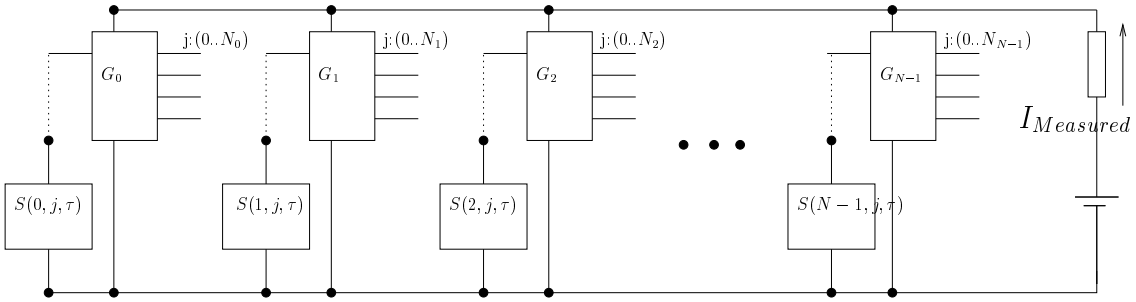


Figure 1.6: Power Consumption Model.

1.3 Side-Channel Attacks

Side-Channel Attacks are very similar to Spectroscopy(NMR) used over the years. While in NMR spectroscopy, patterns in the light spectrum are used to detect the presence of atoms and its environments in an unknown substance, in a Side-Channel Attack the cryptanalyst looks for patterns in the power consumption or EM emission to detect the unknown key value. Apart from cryptanalysis, side-channel could be of great utility to an electrical engineer, as a probing tool more in line with spectroscopy used in physics.

1.3.1 Modelling the Side-Channel

In this manuscript we consider the Power Traces $S(I_i, I_{i-1}, t)$ of a combinatorial circuit. That is it has no memory. I_i and I_{i-1} are the i^{th} and $(i-1)^{th}$ input to this block, and t is the time sample of the measured current from power supply. Power is only consumed when the input goes through a transition. The power trace is recorded for a time duration [0,T] where the transition is applied at t=0 and T is any time after that when there is no further change in measured current. In this manuscript we use a recursive model of power consumption. For purpose of illustration, we consider a block with N subblocks.

A block with N inputs is characterized by its step current response as the input vectors undergoes a transition $i_j \rightarrow i_k$. Note that output loads are considered to be part of this block.

$$S^{i_j \rightarrow i_k}(\tau) = I_{measured}(\tau) \tag{1.1}$$

and the block is characterized by the set of all step responses corresponding to each transition.

$$\bigcup_{\substack{0<i<2^N-1 \\ 0<j<2^N-1}} S^{i_j \rightarrow i_k}(\tau)$$

Now we view the DUT as recursively organized in various subblocks. At the topmost level the DUT consists of N such gates(see figure. 1.6) where k_{th} gate has N_k possible input transitions at it's input. That is the length of the input transition alphabet to the k_{th} gate is denoted as N_k. Furthermore

- $S(k, j, t)$ denotes the step current response $s(\tau)$ associated with j_{th} transition of the k_{th} gate.
- $A(k, j, I_i, I_{i-1})$ denotes the jth transition on the kth block is activated, during the interval depending on I_i and I_{i-1}

Next we normalize the traces to have a zero mean. So we will assume that $S(k, j, t)$ has a zero mean from now onwards. We even redefine the activation function as:

$$\begin{aligned} T(k, j, I_i, I_{i-1}) &= 1 \quad \text{if input vector to the kth gate} \\ &\qquad\qquad \text{undergoes jth transition} \\ T(k, j, I_i, I_{i-1}) &= -1 \quad \text{if input vector to the kth gate} \\ &\qquad\qquad \text{remains unchanged} \\ T(k, j, I_i, I_{i-1}) &= 0 \quad \text{if } j > N_k \end{aligned}$$

Note that $A(k, j, I_i, I_{i-1})$ can be written in terms of $T(k, j, I_i, I_{i-1})$ as

$$A(k, j, I_i, I_{i-1}) = \frac{1}{2} \times (1 + T(k, j, I_i, I_{i-1})) \tag{1.2}$$

The advantage of this representation is that we can write, for M random input transitions

$$\sum_{i=0}^{M-1} T(k_1, j_1, I_i, I_{i-1}) \times T(k_2, j_2, I_i, I_{i-1}) = 0 \quad \text{if} k_1 \neq k_2 \& j_1 \neq j_2$$

$$\sum_{i=0}^{M-1} T(k_1, j_1, I_i, I_{i-1}) \times T(k_2, j_2, I_i, I_{i-1}) = M \quad \text{if} k_1 = k_2 \& j_1 = j_2$$

This is based on the assumption that all transitions are independent, and it closely follows the mathematical definition of orthogonality over M random input transitions.

With this notation we can write one power trace for input vector transition from I_i to I_{i-1} as

$$S(t) = \sum_{k=0}^{N-1} \sum_{j=0}^{N_k-1} A(k, j, I_i, I_{i-1}) \times S(k, j, t) \tag{1.3}$$

$$S(t) = \sum_{k=0}^{N-1} \sum_{j=0}^{N_k-1} \frac{1}{2} \times (1 + T(k, j, I_i, I_{i-1})) \times S(k, j, t) \tag{1.4}$$

1.3.1.1 Post-Processing

Now we want to find out the step current response associated with q_{th} transition of the p_{th} gate. For that purpose we apply M random transitions $< I_i, I_{i-1} >$ at the input which also includes transitions that will trigger the event $A(p, q, t)$ and multiply each trace by $T(p, q, I_i, I_{i-1})$. $s_{acc}(t)$ denotes the accumulated trace after this multiplication and

addition.

$$
\begin{aligned}
s_{acc}(t) &= \sum_{i=0}^{M-1} T(p,q,I_i,I_{i-1}) \times S(t) \\
s_{acc}(t) &= \sum_{i=0}^{M-1} T(p,q,I_i,I_{i-1}) \times \sum_{k=0}^{N-1}\sum_{j=0}^{N_k-1} \frac{1}{2} \times (1 + T(k,j,I_i,I_{i-1})) \times S(k,j,t) \\
s_{acc}(t) &= \sum_{i=0}^{M-1} T(p,q,I_i,I_{i-1}) \times \sum_{k=0}^{N-1}\sum_{j=0}^{N_k-1} \frac{1}{2} \times S(k,j,t) + \\
&\quad \frac{1}{2} \times \sum_{i=0}^{M-1}\sum_{k=0}^{N-1}\sum_{j=0}^{N_k-1} (T(k,j,I_i,I_{i-1})) \times T(p,q,I_i,I_{i-1}) \times S(k,j,t) \\
s_{acc}(t) &= \sum_{i=0}^{M-1} T(p,q,I_i,I_{i-1}) \times N \times N_k \times \frac{1}{2} \times E_{j,k}[S(k,j,t)] + \\
&\quad \frac{1}{2} \times \sum_{k=0}^{N-1}\sum_{j=0}^{N_k-1} S(k,j,t) \times \sum_{i=0}^{M-1} (T(k,j,I_i,I_{i-1})) \times T(p,q,I_i,I_{i-1}) \\
s_{acc}(t) &= \sum_{i=0}^{M-1} T(p,q,I_i,I_{i-1}) \times N \times N_k \times \frac{1}{2} E(S(t)) + \frac{M}{2} S(p,q,t)
\end{aligned}
$$

Since we preprocessed the traces $S(t)$ to have a zero mean , we can find out the step current response as

$$s_{acc}(t) = \frac{M}{2} S(p,q,t) \tag{1.5}$$

1.3.1.2 Side-Channel Oscilloscope

In the above section we outlined a method to find the current response associated with the pth transition of the qth gate. This process can continued recursively for the qth gate until we have only a single net, in which case we can derive the voltage waveform from the step current response using basic circuit behaviour. In the above mentioned method, the orthogonality of $T(k,j,I_i)$ functions play a pivotal role. Even in Template Side-Channel Attacks [4], a major step is to find orthogonal representation of the acquired traces. To guarantee this orthogonality we can divide a block recursively into two sub-blocks using minimum cut bisection, and finally arriving at the target transition.

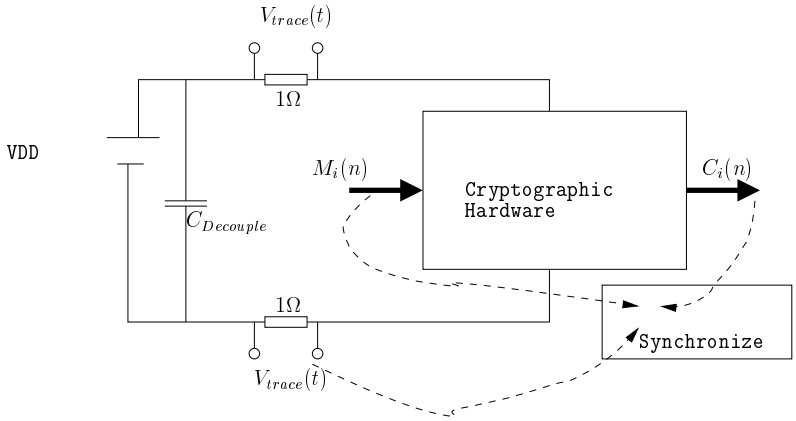
We can see that power analysis techniques can be used for probing each single net behaviour in the circuit [13], thus acting as an oscilloscope even for physically inaccessible components. This will be beneficial for modelling new technologies based on in- circuit measurement. The difference between Side-Channel Attacks(SCA) and Side-Channel Oscilloscope(SCO) is that in SCA, the user does not have the full knowledge of the circuit. So the functions $T(k,j,I_i,I_{i-1})$ are only guesses. In SCO, the user can calculate the activation functions, but for him the unknown is the response of the fabricated circuit.The major assumptions that we made are that the transitions are orthogonal/independent, which may not be true for all circuits, however for some amount of interdependence we still get a magnification for the target transition, and dependent transition current response remains present as noise. This process can be further improved by imposing DFT rules, and using

more complicated post processing techniques such as Principal Component Analysis as used in Template Attacks [4].

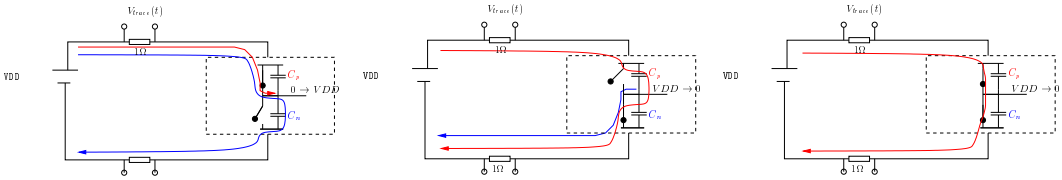
1.3.2 Classification

Side-Channel Attacks are carried out in two steps. First the side channel traces are acquired during the encrypt/decyrpt operation in the cryptographic hardware. Next this trace database is analyzed using statistical signal processing techniques. Based on the acquisition method we can classify side channel attack into two broad domains:

1.3.2.1 Power Side Channel



(a) DPA Setup



(b) $Transition 1 \rightarrow 0$ (c) $Transition 0 \rightarrow 1$ (d) *Short Circuit* and *leakage* Current

Figure 1.7: Power Side Channel

Figure. 1.7 depicts the power side channel. In Fig. 1.7(a) we see the schematic diagram of a power trace acquisition setup. The synchronizing element is implemented in a PC or in the oscilloscope, which sends known messages, stores the outputs ciphers and also triggers the acquisition of the V_{trace}.

In figure. 1.7(b) we see C_P is discharging and C_N is being charged by current drawn from the power supply, which is visible from the V_{trace} taken across the very small resistance in the V_{DD} line. In figure. 1.7(c) we see the complementary action where C_N is discharging through the GND line, and C_P is being charged through the V_{DD} line. C_P and C_N denotes respectively the load capacitance w.r.t the Power line and w.r.t the Ground Line.

Figure.1.7(b) shows the path of the short circuit current and the leakage current through the circuit. While the short circuit current is dependent on computing transitions(thus leaks information), the leakage current is static.

1.3.2.2 EM Side Channel

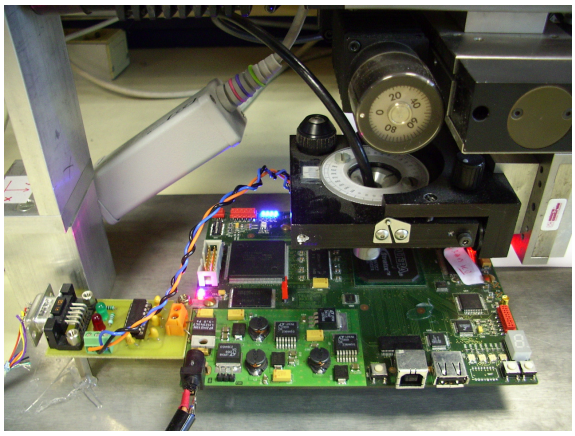


Figure 1.8: EM Acquisition.

Figure. 1.8 shows the EM acquisition setup used in our lab. A precisely controlled antenna(μm) is placed very close to the circuit under attack. This antenna setup is automated to acquire traces at various positions over the chip, with a synchronization similar to fig. 1.2(a).

1.3.2.3 Correlation Attacks

Correlation Power Analysis (*CPA*) and Differential Power Analysis (*DPA*) can be easily understood from the Side-Channel Model described in subsection 1.3.1. In subsection 1.3.1, we have assumed that the Activation Function $A(k, j, I_i, I_{i-1})$ is known to the analyst. For an attacker, this is not true, since he/she is not in possession of the key value. So he/she makes an exhaustive set of guesses depending on the target key bits. She has a pre-defined model of the step response power supply current(denoted as $S(\tau)$ in subsection 1.3.1) which is a impulse charging/discharging current in case of CMOS circuits. The guess which comes closest to this expected response is the right key guess.

In DPA the set of power traces is divided into two sets using the partitioning function for each key guess j and trace $S_i(t)$. The right guess will result in a visible peak(impulse charging/discharging current) when the set with no activity S_0 is subtratced from the set with probable activity S_1.

$$\begin{aligned} S_0 &= S_{ij}|D(.,.,.) = 0 \\ S_1 &= S_{ij}|D(.,.,.) = 1 \end{aligned}$$

In DPA terminology, the function $D(.,.,.)$ is known as a *bias function*. An example of such a bias function for *DPA* on DES is (taken from reference [28])

$$D(C_1, C_6, K_{16}) = C_1 \oplus SBOX1(C_6 \oplus K_{16})$$

where

- C_1 is 1 bit of the ciphertext that is XORed with 1st bit of 1st S-Box.
- C_6 is the 6 bits of the ciphertext that is XORed with subkey K_{16}.

- K_{16} is the 6 bits of the 16th round subkey feeding into S-Box 1.
- SBOX1(x) is a function returning the first bit resulting from looking up x in the 1st S-Box.

CPA is a more advanced form of *DPA*. In the above mentioned *DPA* example we have targeted only one transition of the S-Box output. To increase the efficiency we often target a set of transitions. Let's say we're targeting k equipotentials in the circuit whose states before and after the transition are S_i and S_{i+1}. Assuming CMOS logic circuit the power consumption associated with this transition will be the *Hamming weight* difference of the states. So the relationship between current consumption(W) and this transition can be written as

$$W = aH(S_i \oplus S_{i+1}) + b$$

Where H denotes the *Hamming weight*, a denotes a scaling constant, and the power consumption of the rest of the chip is denoted by a constant b. Note that we assume here implicitly that power consumption of the rest of the chip is noway related to out target transition, which gives rise to this linear relationship.

Next we should choose the key among the key guesses whose power consumption(W) correlates best with the *Hamming weight* guess $H(S_i \oplus S_{i+1})$. For that we define the correlation metric as

$$\rho_{WH} = \frac{cov(W, H)}{\sigma_H \sigma_W}$$

over a set of known input messages to the cipher. The guess with highest ρ_{WH} is the correct guess. For a more detailed explanation please see [8].

1.3.2.4 Template Attacks

Principal Component Analysis Principal Component Analysis(PCA) is a mathematical tool to identify patterns in a n-dimensional dataset. It transforms and represents the n-dimensional dataset, in terms of n orthogonal feature vectors, classified from major features to minor features. Major applications of PCA are in image compression, where the dimensions with smaller features(variation) are ignored leading to lossy compression, and blind deconvolution techniques (Given a signal trying to find out the the original signals of which the given signal is the convolution). We describe it here briefly.

- Model the data as an n-dimensional dataset. For example a terrain can be represented as a three dimensional dataset.
- Preprocess to have a zero mean in each dimension. That is, subtract $\overline{x}$ from all the x values, $\overline{y}$ from all the y values and so on.
- calculate the covariance matrix. For example, the covariance matrix of a 3 dimensional (x, y, z) dataset is

$$C = \begin{bmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z) \end{bmatrix}$$

  Note that the covariance matrix is symmetrical since $cov(a, b) = cov(b, a)$, and for a n-dimensional dataset we need to calculate $\frac{n!}{(n-2)!\times 2}$ covariances.

- Find out the normalized eigenvectors(and eigenvalues) of the covariance matrix. These eigenvectors are what represent the data best, from eigenvectors with the highest eigenvalue to lowest eigenvalue. The eigenvector with the highest eigenvalue is called the *Principal Component* of the dataset.

  Once we have sorted the eigenvectors from highest to lowest, we get the components in their order of significance. Now if we neglect the dimensions with low significance, we loose some information, but it may be enough for our purpose. This new dataset with only few principal dimension is known as *principal subspace* of the dataset.

Now we illustrate, how a *Template Attack* is carried out as in reference [4]. The idea is to build templates(Principal Subspaces) for key bits entering the first S-box in AES. For each key candidate

- Feed the first S-box with a deterministic sequence of plaintexts (a counter).
- Feed all other S-boxes with random inputs.
- Acquire the power traces for the above operations. A power trace with N time samples is considered as a N-dimensional vector. In general we take the average trace of deterministic sequence used for each key. This is because in CMOS logic, the power consumption depends on the change of state of the circuit and not on the actual state.
- Now we construct a principal subspace of the above mentioned N-dimensional dataset (say with M feature vectors where $M < N$) denoted as $TEMPLATE_{K\times M}$. Now our Template is built. K is the number of key candidates.
- To carry out the attack, we use the same deterministic sequence and acquire the power trace $T_{unknown}(n_1, n_2 \dots n_N)$, but now we don't know the key.
- Now we transform the acquired N-dimensional power trace to M-dimensional principal subspace $T_{unknown}(n_1, n_2 \dots n_M)$ by using the retained feature vectors. We do an exhaustive search by comparing $T_{unknown}(n_1, n_2 \dots n_M)$ to the $M-dimensional$ vectors in $TEMPLATE_{K\times M}$ dataset to find out the unknown key.

In actual experiments carried out in [4] $k = 10$, $N = 500,000$, and $M = 20$ for a success rate of 86.7%. We can see raw calculation of the above method will be computing intensive, reference [4] addresses the issue of minimizing these computations.

1.3.3 Counter Measures

The reported counter-measures to side-channel attacks can be classified as dynamic counter-measures(incorporated at run-time) masking and static counter measures(incorporated at design time) or hiding.

1.3.3.1 Dynamic or "Masking"

The principle of dynamic counter-measures is to decorrelate computing form the power supply current, by randomizing transitions, commonly known as "masking". This can be either precharging signals with random values, or introducing random delays in computing paths [33, 2, 43]. This masking techniques are introduced at the algorithmic level. Details of implementing and attacking such countermeasures can be found in [34, 40].

Random Switching Logic (*RSL*) *RSL* proposed by Suzuki [45], in which the actual data is masked with a pseudo-random Mask bit. Let's say the actual data is a and b, with their masked counterparts x and y, and our psuedo random mask bit is r.

$$x = a \oplus r \qquad y = b \oplus r$$

An *RSL* NAND and NOR gate have the following functionalities:

$$Z_{NOR} = \overline{(x.y + (x+y).\overline{r})} \qquad Z_{NAND} = \overline{(x.y + (x+y).r)}$$

When $r = 0$

$$x = a \qquad y = b$$

$$Z_{NOR} = \overline{(x+y)} \qquad Z_{NAND} = \overline{(x.y)}$$

When $r = 1$

$$x = \overline{a} \qquad y = \overline{b}$$

$$Z_{NOR} = \overline{(x.y)} = \overline{x} + \overline{y} \qquad Z_{NAND} = \overline{(x+y)} = \overline{x}.\overline{y}$$

In an *RSL* circuit, only the primary input signals are masked, and the mask is removed at primary outputs. Internal signal nodes remain in masked mode while traversing from one gate to another gate. *RSL* gates also have a *enable* to avoid glicthes. To summarize

- *RSL* executes masked operations using the same 1 bit random value.
- *RSL* executes operations only when the enable signal is 1, otherwise drives 0.
- *RSL* countermeasure will be successful if
  - The transition of the random signal r is not biased
  - The *enable* signal rises after all other input signals are stable(to avoid glitches)
  - As noted by [39] the random mask bit puts the circuit into one of two dual configurations, loading imbalances between the routes of these two configurations will leak information.

Path Switching Logic In Path Switching Logic [18] the routing wire imbalances are balanced in a dynamic fashion, as explained is figure 1.9 the `true` and `false` signals of a dual-rail logic takes alternate route from source to sink. If this path switchin is done in an pseudo-random manner, this can reduce the effect of loading imbalance in side channel leakage and thus enhance a logical level counter measure such as RSL or WDDL. This is pretty easy to implement in an FPGA with partial or run-time reconfigurable capability.

1.3.3.2 Static or "Hiding"

Static Countermeasures rely on producing a constant power consumption profile independent of the data being computed. This is commonly done with differential signaling with a precharge to '0', where power consumption profile of one rail masks that of the other. Examples of such countermeasures are WDDL [46], DPL: DualRail Precharge Logic [14], STTL [36, 41], MDPL [35]. 1-out-of-n Asynchronous signaling also falls in this category.

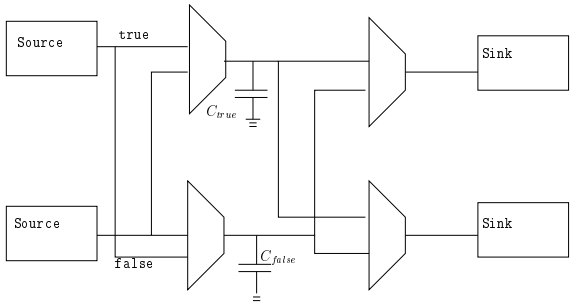


Figure 1.9: Path Switching Logic.

Wave Dynamic Differential Logic(*WDDL*) *WDDL* is a dual-rail logic with precharge where logical values are represented by

$$logic-0 \Rightarrow (1,0) \qquad logic-1 \Rightarrow (0,1) \qquad Precharge \Rightarrow (0,0)$$

WDDL gates are implemented with complementary logic:

$$NOR : (z,\overline{z}) = (\overline{x}.\overline{y}, x+y) \qquad NAND : (z,\overline{z}) = (\overline{x}+\overline{y}, x.y)$$

- To precharge every signal to $(0,0)$ the primary inputs are forced to 0(i.e $(x,\overline{x}) = (0,0)$). These precharge propagates as a wave thorugh the circuit.
- For inversion differential pairs are twisted.
- For each cycle there are the following transitions independent of the computation.

$$[(1,0)|(0,1)] \rightarrow (0,0) \qquad \text{and} \qquad (0,0) \rightarrow [(1,0)|(0,1)]\,.$$

In *WDDL* evaluation phase from (0,0), the *OR* gate evaluates faster than the *AND* gate, since *OR* switches it's output with a single transition on it's input, whereas the *AND* gate waits for two transitions. This is known as the **Early Evaluation Fault** [142]. More the success of *WDDL* calls for balanced interconnect throughout the circuit.

Backend Duplication Two papers address this issue of balancing interconnects: the "*fat wires*" routing [17] and the "*backend duplication*" method [15]. These methods are presented on dual-rail circuits, but can easily be generalized to any bus width. As illustrated in Fig. 1.10, the fat wires equitemporals make a $\pi/4$ angle with the cell rows, whereas the backend duplication equitemporals are collinear with the (x,y) vector, where x is the routing *pitch* and y is the placement *cell height*.

Masked Dual-Rail Precharge Logic (*MDPL*) *MDPL* is a combined approach towards SCA countermeasure, which is a result of combining *RSL* and *WDDL* described above. An interesting evaluation of this approach can be found in [39] which states "glitches" and "routing imbalance" as two major problems.

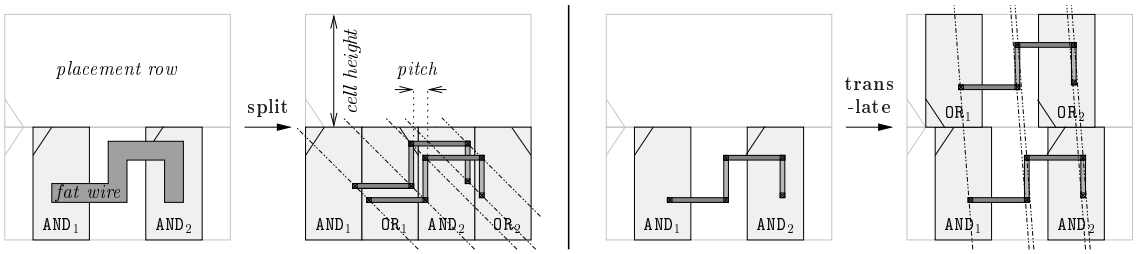


Figure 1.10: Fat wire (*left*) and backend duplication (*right*) paths balancing illustration in semi-custom ASICs using WDDL (logic in which `AND` & `OR` gates are mutually dual.)

Dynamic Current Mode Logic Styles In conventional CMOS logic, power is consumed only for $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions, and no power is consumed for cycles following evaluation $0 \rightarrow 0$ and $1 \rightarrow 1$. This results in data-dependence of the power consumption. On the contrary Dynamic Current Mode Logic styles consume power for all 4 transitions. Several authors have investigated the possiblity of using Dynamic Current Mode Logic for counter-Measures against SCA [24]. Another variation of this approach is Sense amplifier Based Logic(*SABL*) [48].

Current Mask Generation (*CMG*) Current Mask Generation [27] is a static countermeasure which uses current loop-feedback technique to make the power consumption appear constant. The power supplys for cryptographic cells are connected to the output of this current mask generator instead of *VDD* and *GND*. The *CMG* is an analogic block which consists of *current mirror* and a *voltage follower* used as a current feedback circuit. For more details plese look into reference [27].

Asynchronous Circuits Asynchronous Circuits have several desirable charecteristics as a SCA countermeasure, some of which are "Glitch-free Operation", "Power Constant Signalling", "Randomization of switching instants" etc. Several authors have implemented asynchronous countermeasures in ASIC We will discuss more about this technique, and an asynchronous FPGA implementation in the following chapters.

Chapter 2

FPGA Architectures and CAD

2.1 Logic Architecture

The basic logic block architecture often called CLB(*Configurable Logic Block*) for general purpose computing is shown in figure 2.1(a) which consists of 4-input LUT, to implement any four variable boolean function, and a edge-triggered flip-flop, to implement sequential circuits. Commercial FPGAs have a lot of added functionality over this basic logic block, and they are often organized hierarchically as clusters(see fig. 2.1(b)) of these basic logic blocks. Some of these added functionalities are fast carry chain, register cascading inside a cluster, PRESET/RESET functionality for FFs, implementation of synchronous RESET etc.

Apart from general purpose computing, FPGA Logic also includes Hard Macros such as Multipliers, DSP blocks, RAM. Figure 2.2 shows the position of hard macros in island style FPGAs.

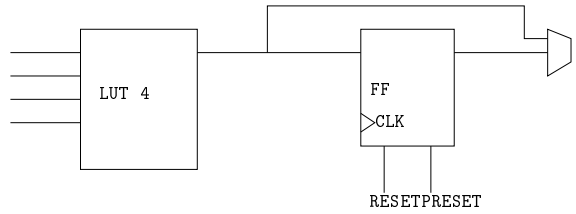
2.1.1 Key Parameters: Logic Architecture

Cluster Size(N) The number of logic elements in the cluster. e.g the Stratix II FPGA from Altera has N=8 [77]. This cluster size is determined after a trade-off analysis between various factors. Some of these factors are: amount of inter-cluster and intra-cluster routing area, LUT architecture etc.

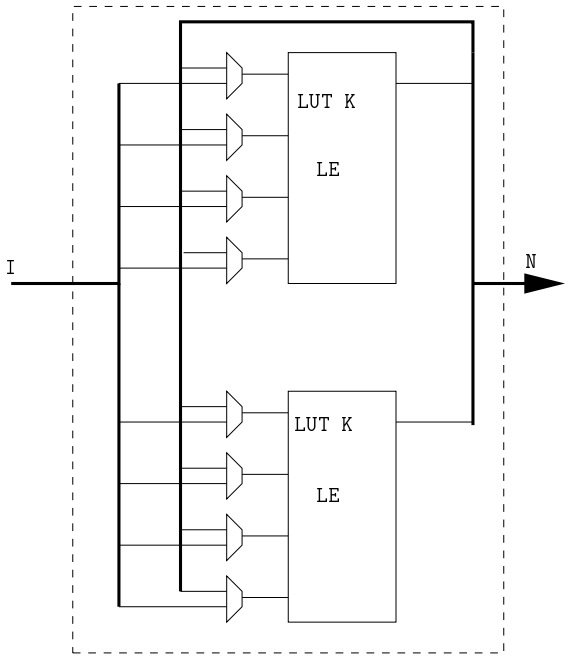
Cluster Inputs(I) Only a fraction of the routing tracks are connected to the cluster. I denotes the number of inputs to the cluster from the inter-cluster routing channels. e.g. in stratix II I=38 [77].

LUT Size(K) The LUT size is determined primarily based on logic depth achieved, and the area of LUT. According to previous studies [54] a LUT Size of 6 gives a optimum performance between speed (Logic depth) and area. In modern FPGAs complicated LUT architectures are used, e.g. in Stratix II a fracturable LUT Mask (FLM) is used. A FLM LUT is denoted by two parameters K,M. K denotes the size of the largest LUT, and M denotes the number of extra input pins that are brought into the LE to achieve the largest LUT size. In Stratix II [77], the basic LUTs are of size 4, but a 6-LUT can be composed from them using 2 more inputs (i.e a 6,2 FLM LUT).

According to previous research [54] $2N + 2$ inputs per cluster is sufficient to achieve 98% logic utilization.



(a) Logic Element.



(b) Logic Cluster.

Figure 2.1: Logic Architecture.

A new research direction in Logic Block design is shadow clusters [70], where a cluster is added with every hard macro in a heterogeneous FPGA. In case the hard macro is not used, the cluster uses efficiently the routing resources, meant for the hard macro. Another new method of better utilization of FFs is to add a functionality in the LE so that LUTs and FFs can be used separately, leading to higher logic density [68].

2.2 Routing Architecture

2.2.1 Island-Style FPGAs

Figure 2.2 describes the Island style FPGA architecture. Each CLB connects to the routing fabric via Connection Boxes(`CBOX`). Each Connection Box is laid over X and Y directed segments(Routing Tracks). At each Switch Box these X and Y directed segments can be connected to each other in a pre-defined fashion (described later in subsection 2.2.1.2. CLBs or Configurable Logic Blocks are marked as `CLB` in figure 2.2. Often to increase performance, major FPGA vendors include pre-defined MACROs in the FPGA architecture(such as Multipliers, DSP blocks). The position of such a MACRO is marked in figure 2.2. Note that the Hard Macro replaces CLBs to fit into the routing architecture.

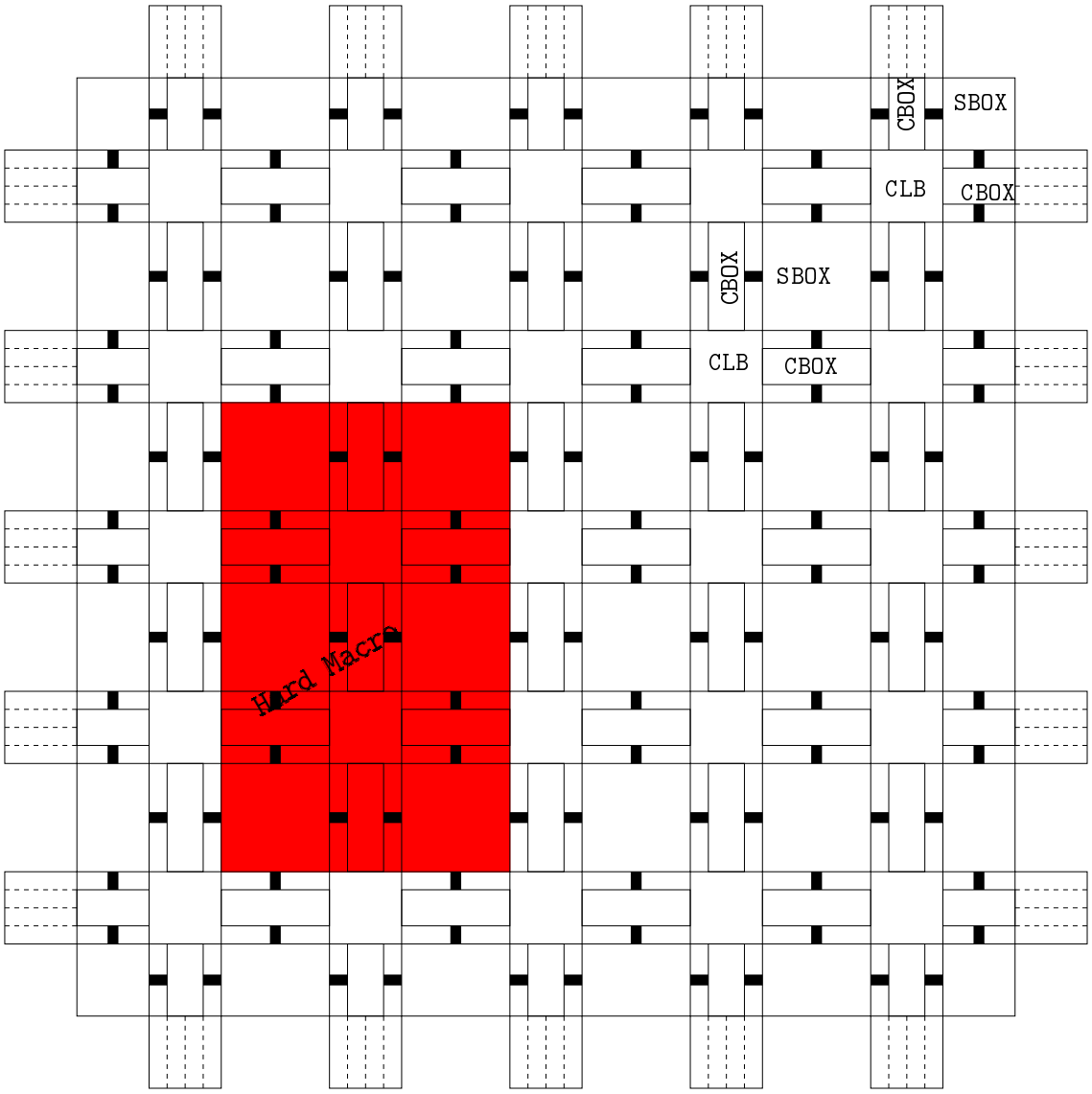


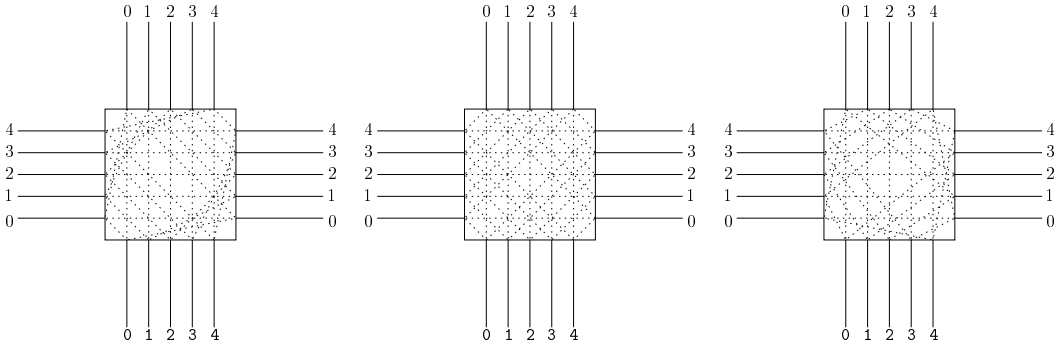
Figure 2.2: Island Style FPGAs.

2.2.1.1 Key Parameters

Connection Box Flexibility(*Fc*) The number of segments to which each CLB pin connect, in a connection box. (Expressed as a fraction of number of tracks).

According to previous Research [87], for a good routability Fc should be equal to at least 50%. *Fc* could be different for different types of pins. For example in VPR we can specify Fc_{input},Fc_{output},Fc_{pad}.

Switch Box Flexibility(*Fs*) Corresponds to the number of segments to which an input to the switchbox can connect. According to [87], for *Fs* greater than 2, we don't see a major improvement in routability.



(a) Disjoint. (b) Universal. (c) Wilton.

Figure 2.3: Switch Boxes.

Segment Frequency Fraction of segments having the same length, and other parameters. e.g: Xilinx 4000x series FPGAs use 25% length-1 segments, 12.5% length-2 segments, 37.5% length-4 segments, and 25% segments with length equal to $\frac{1}{4}th$ of the FPGA.

Segment Length The no. of CLBs covered by a segment.

Connection Box Population(*frac_ cb*) The number or percentage of CLBs that can connect to a segment which spans multiple CLBs. Not to be confused with *Fc*, *frac_ cb* is a property of a segment and not that of a connection box. For example, a segment with *frac_ cb* equal to 60%, and which spans 5 CLBS, only 3 of those 5 can connect to the segment.

Switch Box Population(*frac_ sb*) For a segment spanning multiple CLBs, the number or percentage of switch box positions where it can have a switch. For example, *frac_ sb*) could not be less than 2, since it should have at least two switches at its ends.

2.2.1.2 Switch Boxes

Switch box is one of the critical components in the routing architecture. We can see the option of these switch boxes inside the routing fabric from figure 2.2. Figure 2.3 describes three major types of switchboxes in FPGAs.

Disjoint Disjoint or subset switchbox 2.3(a) is the simplest switch box where the i_{th} track from any direction can only connect to i_{th} from other sides. A major disadvantage of subset is that it creates routing domains inside the routing fabric. It divides the routing fabric into W disjoint sets where W is the no. of tracks in the channel.

Wilton Wilton Switch Box is an evolution from the subset switchbox, where every diagonal connection undergoes a rotation/anti-rotation. Because of this feature it doesn't create any domains in the routing fabric.

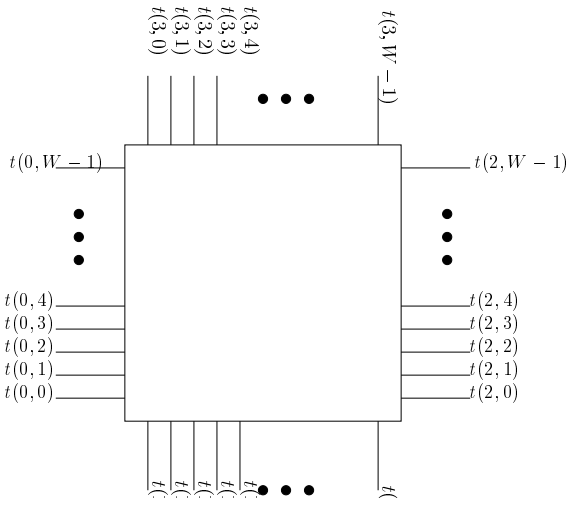


Figure 2.4: Representation of a Switch Box.

Universal Any switchbox satisfying the "Universality" criteria is called a universal switchbox. Any switchbox with N sides and W terminals on each side is universal, is any group of nets satisfying dimensionality constraint(i.e the number of nets routed through each side of the switchbox can not exceed W) is simultaneously routable through the switchbox [56].

2.2.1.3 Mathematical Representation

We will use the following notation [91] to describe switchboxes: each terminal is represented as $t(j,i)$, where j denotes each subset corresponding to each side ($0 =$ left, $1 =$ top, $2 =$ right, $3 =$ bottom) and $i \in [0, W[$ denotes the position of the terminal in that subset. For example, a $W \times W$ "subset" switchbox $\mathbf{S}$ can be represented as a set of arcs between these terminals:

$$\mathbf{S} = \bigcup_{i=0}^{W-1} \left\{ \begin{array}{c} [t(0,i), t(2,i)], \\ [t(1,i), t(3,i)], \\ [t(0,i), t(1,i)], \\ [t(1,i), t(2,i)], \\ [t(2,i), t(3,i)], \\ [t(3,i), t(0,i)]. \end{array} \right\}$$

A $W \times W$ "universal" switchbox $\mathbf{S}$ can be represented as a set of arcs between these terminals:

$$\mathbf{S} = \bigcup_{i=0}^{W-1} \left\{ \begin{array}{c} [t(0,i), t(2,i)], \\ [t(1,i), t(3,i)], \\ [t(0,i), t(1,W-i-1)], \\ [t(1,i), t(2,i)], \\ [t(2,i), t(3,W-i-1)], \\ [t(3,i), t(0,i)]. \end{array} \right\}$$

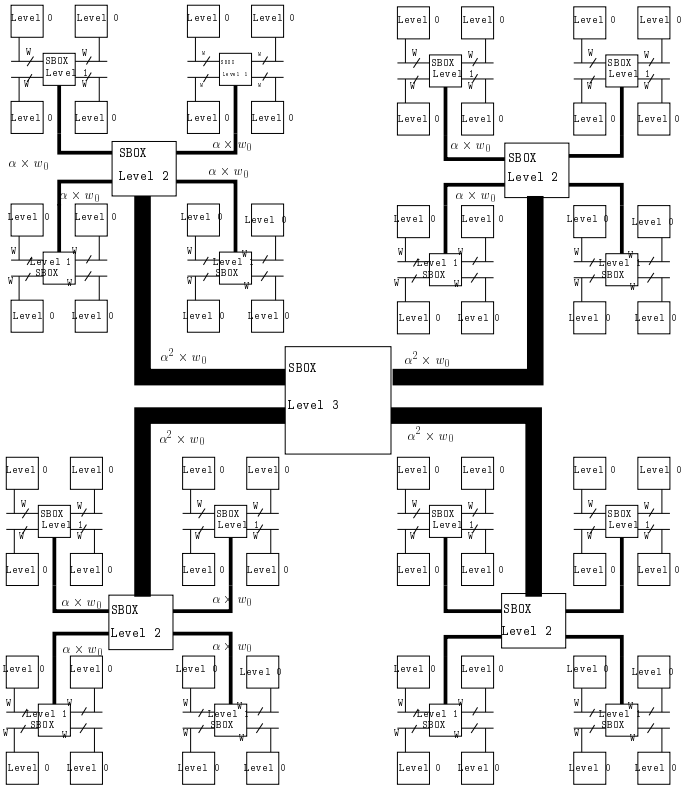


Figure 2.5: Generic Hierarchical Gate Array

A $W \times W$ "wilton" switchbox $\mathbf{S}$ can be represented as a set of arcs between these terminals:

$$\mathbf{S} = \bigcup_{i=0}^{W-1} \left\{ \begin{array}{c} [t(0,i), t(2,i)], \\ [t(1,i), t(3,i)], \\ [t(0,i), t(1,(W-i) \bmod W)], \\ [t(1,i), t(2,(i+1) \bmod W)], \\ [t(2,i), t(3,(2W-2-i) \bmod W)], \\ [t(3,i), t(0,(i+1) \bmod W)]. \end{array} \right\}$$

2.2.2 Hierarchical Gate-Arrays

In recent times hierarchical gate array or Butterfly Fat Tree structures have generated considerable interest in the research community. Notably by [61, 80]. In this type of routing architecture the logic blocks are organized in a hierarchical fashion. The main advantage of using this routing fabric is that, the point to point connections between logic blocks goes through fewer number of switches than in the Manhattan grid. However this research is only at its preliminary stage and hence there is no standard terminology to describe detailed routing architecture. Figure 2.5 describes the basic parameters of this architecture.

2.2.2.1 Key Parameters

Arity The number of branches at each level.

Bifurcation Ratio The ration of channel width of a level to its next level. In figure 2.5 the butterfly fat tree has an arity of four and the bifurcation ratio is denoted as α. The logic blocks or clusters are denoted by level 0.

2.3 CAD for FPGA

Given the size of modern FPGAs, which might contain thousands of logic elements and even more programmable routing switches, design automation is an inevitable step to map netlists onto FPGAs. In figure 2.6 the major steps in this process are illustrated. The reader might notice that it is considerably simpler than that of an ASIC design flow, which one of the main reasons of FPGAs' popularity. Commercial FPGA vendors provide an integrated environment to facilitate the execution of these steps. Examples are ISE from Xilinx, or Quartus from ALTERA. However for research purpose we use open source tools VPR and V-PACK.

2.3.1 VPR

Versatile Place & Route (VPR) is a open source software developed at University of Toronto with Vaughn Betz as the Principal author. Apart from providing the capabilities as described in figure 2.6, VPR is also an architecture exploration tool. A wide variety of Island Style Architectures can be specified as an architecture description file, and benchmarks packed with VPACK can then be used to evaluate those architectures. For this purpose all we need is the netlist description in *.blif* format. Newer versions of VPR(5.0) also supports a flow with verilog netlists, and heterogeneous architectures with hard macros.

The Industrial tools such as Altera's FMT(FPGA Modelling Toolkit) and SMT(Synthesis Modelling Toolkit) is also based on VPR [77]. The details of underlying algorithms in VPR can be found in [54]. In this manuscript, we will describe briefly the algorithms, and modifications made to them in later sections.

2.3.2 Architecture Generation

Several architecture parameters that we discussed in the previous section can be tuned using VPR. Figure 2.7 explains the internal data structure used by VPR for placement and routing. Because of this generic graph data structure VPR can perform placement and routing over a wide range of architectures, once the graph is properly generated. This allows an exploratory approach towards architecture selection.

2.3.3 Placement

VPR placement algorithm is based on *Simulated Annealing*. Simulated Annealing is multivariate optimization method first proposed by Kirkpatrick [72], and analogous to the physical annealing process of growing crystals.

In a very high temperature, the atoms in a physical system are in chaos, which will facilitate the restructuring of the material. As we decrease the temperature, the atoms will tend to settle in their minimum energy configurations, however if the temperature is decreased very fast, the atoms will settle into a local minimum energy configuration, instead of the global minimum, leading to defects in the crystal. For this reason, temperature is decreased very slowly, so that the atoms get enough time to go the global minimum energy state.

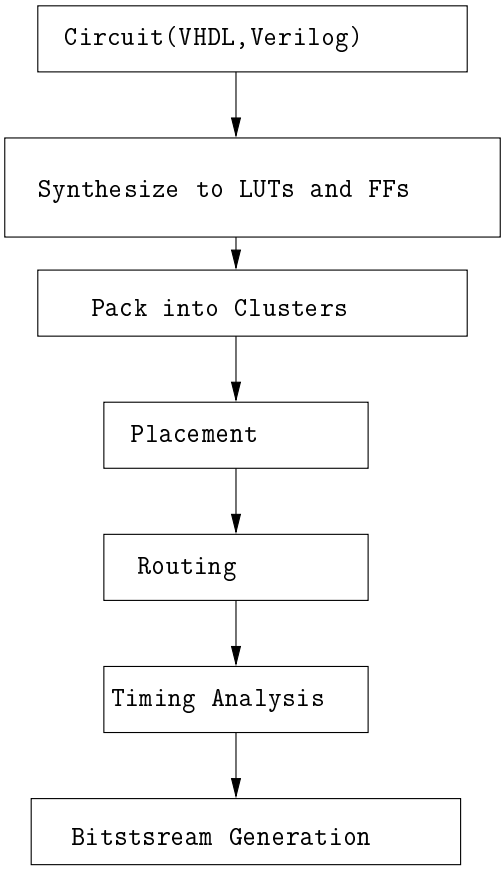


Figure 2.6: Simplified CAD FLow for FPGAs

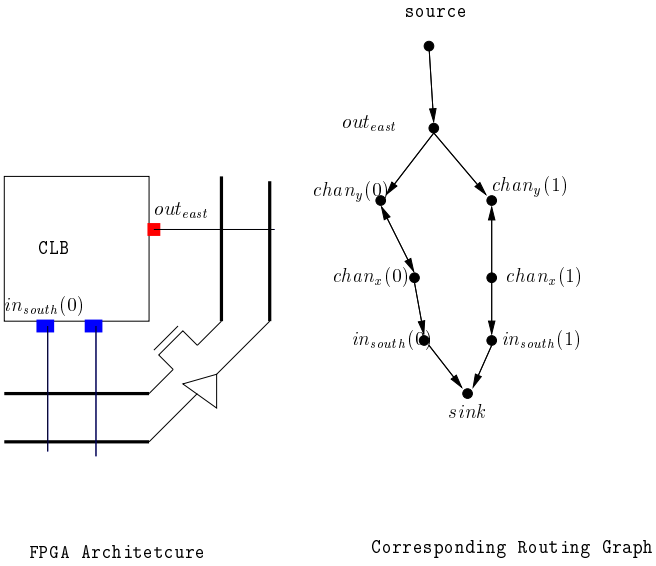


Figure 2.7: Mathematical Representation of FPGA Architecture

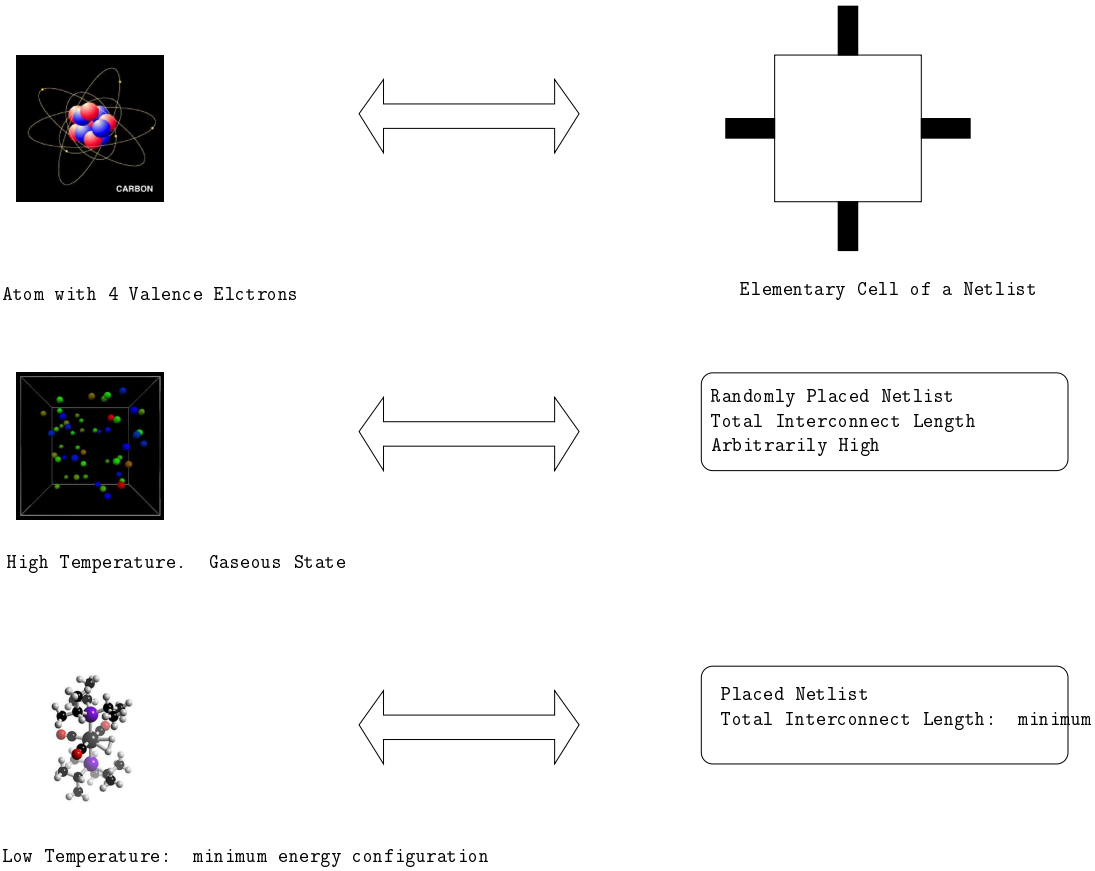


Figure 2.8: Simulated Annealing Placement Analogy

This behavior is simulated in an optimization problem, where the goal of optimization is to minimize a cost function analogous to energy. When the temperature is high, the constituting variables can be sweeped over a large range. This range is decreased each time the cost function decreases. As opposed to greedy algorithms, a new state is accepted only if its cost is less than the previous state, in SA a decrease in cost function is accepted with high probability, and increase is accepted with low probability. This is done to avoid freezing of the system to local minima. Figure 2.8 explains this analogy, and figure 2.9 explains the particular annealing schedule used in VPR.

The cost function used in VPR to minimize interconnect length is

$$Cost_{wiring} = \sum_{i=1}^{N_{nets}} q(i) \times [bb_x(i) + bb_y(i)]$$

Where $q(i)$ is a empirically determined constant which is equal to 1 for nets with less than 3 terminals, and

$$q(i) = 2.79 + 0.02616 \times (N_{terminals} - 50)$$

nets with greater no. of terminals [54].

The cost function can also be determined to minimize the circuit delay. For each connection (i, j) where i is the sink and j is the source. The cost function in terms of

timing is defined as

$$Cost_{timing} = \sum_{\forall i,j \subset Netlist} Cost_{timing}(i,j)$$

The timing cost for each connection is defined as the delay of the connection multiplied by the criticality, which depends on *slack*. The slack of a connection is defined as

$$\begin{aligned} Slack(i,j) &= T_{required}(j) - T_{arrival}(i) - delay(i,j) \\ criticality(i,j) &= 1 - \frac{slack(i,j)}{\text{Max. Delay}\forall(i,j)} \\ Cost_{timing}(i,j) &= delay(i,j) \times criticality(i,j)^{CriticalityExponent} \end{aligned}$$

For more details on the implementation please refer to [54]

2.3.4 Routing

For reader's convenience we briefly recapitulate the basics of a maze router based on Dijkstra's algorithm and Pathfinder negotiated congestion. More information can be found in [81, 53]. The maze router consists of three principal objects

- The Routing Graph: This corresponds to the FPGA interconnection structure. On this graph the partially calculated minimal spanning tree (i.e. the minimum cost upto now to achieve a node from the source) is marked for each net.
- The Priority Queue: A FIFO list of data with priority (i.e. the minimum is always on the top). The nodes on the current wavefront is stored in the priority queue. The priority queue guarantees that the current wavefront always starts expanding from the vertex with minimum cost.
- The Trace Database: An array of linked lists which stores the graph vertices in the proper sequence (source → sink) which defines the routing for a net.

The cost of a node from the source is calculated as

$$\begin{aligned} cost(node) = {} & cost(previousnode) + \\ & cost_{congestion}(node) + \\ & \alpha \times (cost_{depth}(node)) \end{aligned}$$

where α is known as the direction factor and controls the extent of depth-first search.

2.4 Routability Analysis

2.4.1 Rent's Rule

The Rent's rule is based on the works of Mr. E.F Rent in the sixties. His works revealed a remarkable relation between the No. of terminals and the No. of gates in an integrated circuit. Published for the first time by Landman and Russo [75], the Rent's rule is expressed as

$$T = tB^p, \quad 0 \leq p \leq 1, \quad \text{où :} \tag{2.1}$$

- T is the No. of terminals (input or output),

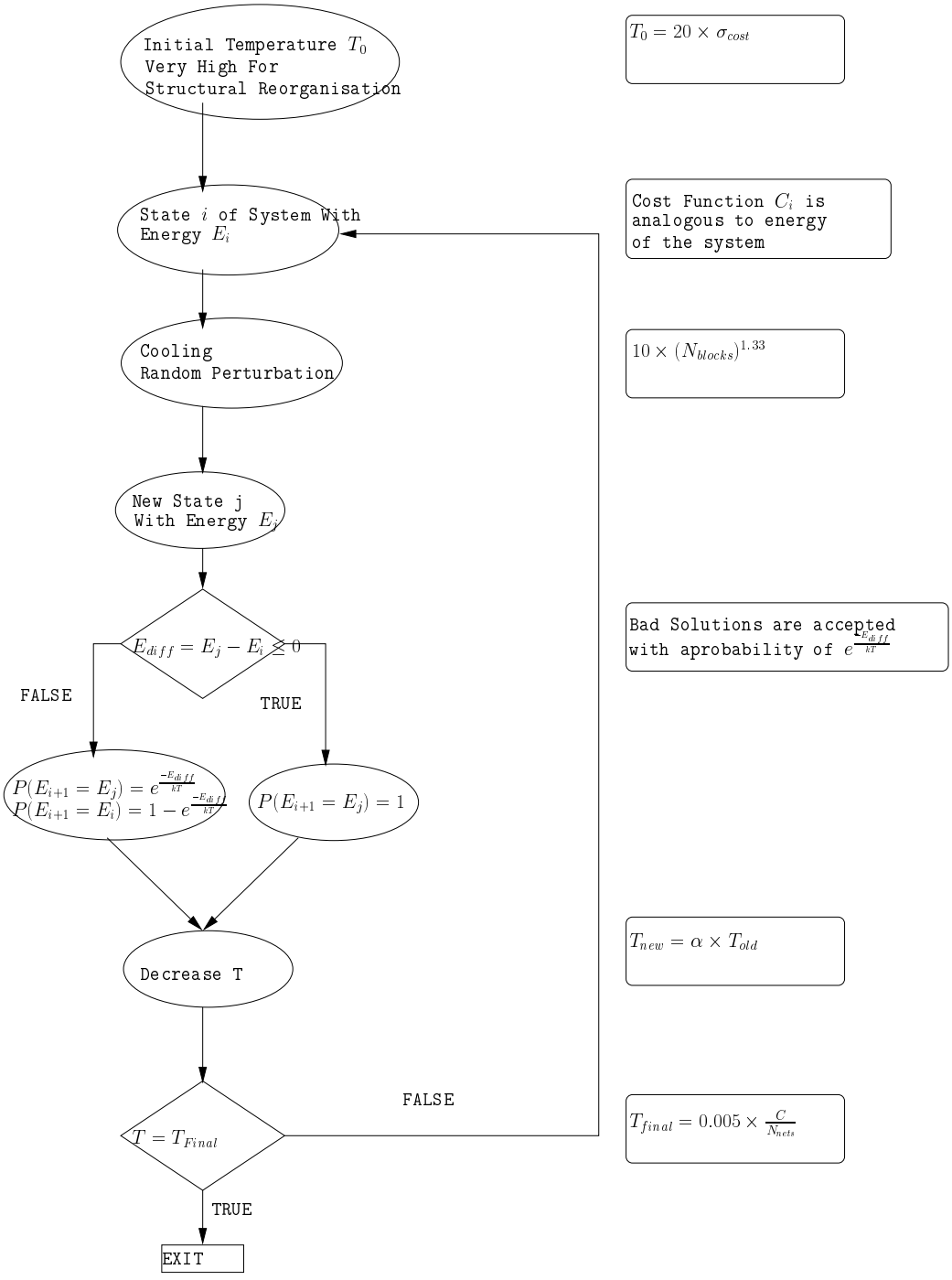


Figure 2.9: VPR Simulated Annealing Schedule

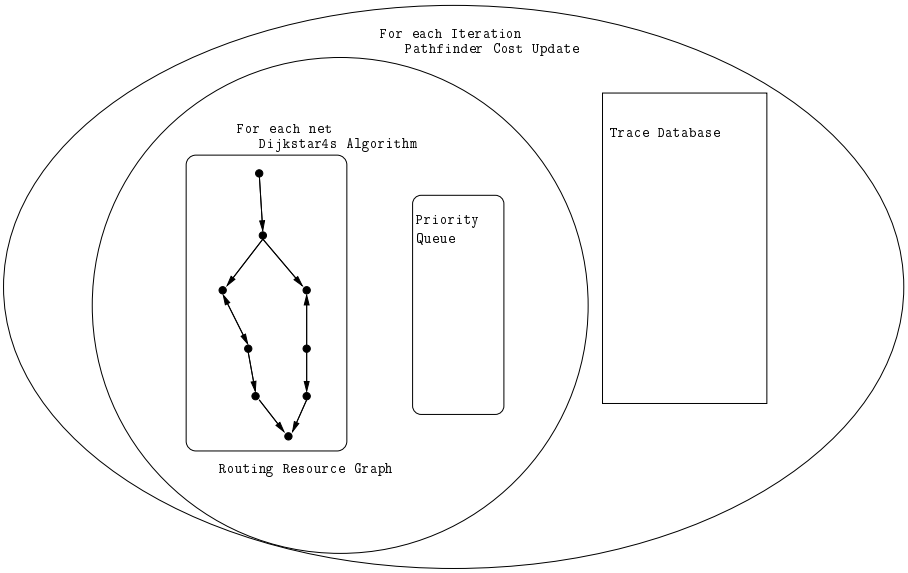


Figure 2.10: PATHFINDER Negotiated Congestion Routing

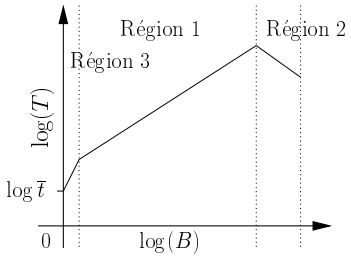


Figure 2.11: Rent plot for a typical netlist.

- B is the No. of blocks in the partition,
- t Rent Coefficient, *i.e.* average No. of terminals per block,
- p is the Rent's Exponent, *i.e.* a measure of complexity of the interconnection network, obtained by recursive bi-partitioning of the netlist graph.

A typical Rent plot shows several zones as shown in Fig. 2.11. These three regions can be represented by three different couplets (t, p). The application of Rent's Rule are estimation of average interconnect length for a given netlist [62, 58] and evaluation of FPGA architecture [86] etc.

2.4.1.1 Definitions

Hypergraph : A hypergraph G is defined as $G \doteq (X, U)$ where X is the set of vertices $(x_1, x_2, x_3, \dots)$ and U is the set of hyperedges $U \subseteq P(X)$, where $P(X)$ represents a set of pratitions of X. In other words a hypergraph is a graph whose edges can connect to more than two vertices. Most of the properties of graphs are also valid for hypergraphs. In this manuscript we will consider each netlist as a hypergraph, whose vertices are the logic gates and the connections ar the hyperedges. This model is particular to electrical circuits, where each logic gate connect to several other logic gates (*fanout*).

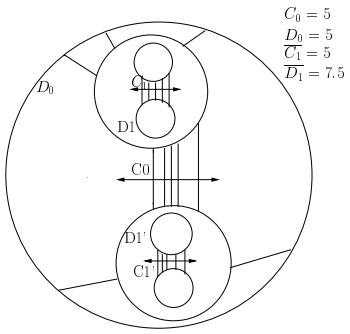


Figure 2.12: Partitioning process of a graph.

Minimum Bisection Let G be a hypergraph whose vertices V are connected by hyperedges E, we note that $G = (V, E)$. The Mincut Bisection of G is the partition of V into two disjoint sets U and W, such that $|U| = |W|$, and $e(U, W)$, the number of hyperedges $\{(u, w) \in E \,|\, u \in U \, et \, w \in W\}$ be minimum [65].

Degree of a partition (D_i) : The degree of a pertition is defined as the number of outgoing and incoming edges of that partition. For example, a partition which contains only one vertex, the degree of the partition is equal to the degree of the vertex. In this thesis manuscript, we denote D_i as the degree of partitions obtained after $i^{\grave{th}}$ recursive partioning.

MinCut (C_i) : The MinCut of a hypergraph is defiend as the number of edges between the partitions obtained through Minimau bisection of the Hypergraph. In this manuscript, we use C_i to denote the value of the MinCut obtained after $i^{\grave{e}me}$ recursive partition.

Locality : We define the locality of a graph as:

$$L_i \doteq \frac{2\,C_i}{D_i}\,, \tag{2.2}$$

where C_i is the value of the MinCut of the graph and D_i is its degree. For example, an isolated partition(graph) has infinite $(+\infty)$ locality, since D_i is 0. In the case of a hypergraph, Locality is defined as the ratio of the number of terminalsconnected to other terminals in its dual partition, and the degree of the hypergraph.

2.4.1.2 Rent Parameters and Locality

The bi-partition of a hypergraph G_0 (respectively G_i) generates two partitions G_1 (resp. G_{i+1}) and $G_1{}'$ (resp. $G_{i+1}{}'$), as described in figure 2.12. To keep the mathematics simple, we consider that our netlist can be represented by a graph, that is each net of the netlist is connected to two and only two terminals. We can than deduce the following relation between D_0 (resp. D_i), D_1 (resp. D_{i+1}) et C_0 (resp. C_i) :

$$D_0 = 2\overline{D_1} - 2C_0\,, \tag{2.3}$$

where $\overline{D_1}$ is the mean of D_1 and $D_1{}'$. Following our partitioning process in a recursive manner, upto indivisible partitions(i.e. Partitions consisting of unque cells), the level which

we denote as K, we can construct the following equations:

$$\begin{aligned} D_0 &= 2\overline{D_1} - 2C_0 \\ 2\overline{D_1} &= 4\overline{D_2} - 4\overline{C_1} \\ 4\overline{D_2} &= 8\overline{D_3} - 8\overline{C_2} \\ &\vdots \\ 2^{K-1}\overline{D_{K-1}} &= 2^K\overline{D_K} - 2^K\overline{C_{K-1}}\,. \end{aligned}$$

From the previous equations we can say that:

$$2^K\overline{D_K} - D_0 = 2\overline{C_1} + 4\overline{C_2} + 8\overline{C_3}\ldots + 2^K\overline{C_{K-1}}\,. \tag{2.4}$$

By replacing C_i by $\frac{L_i D_i}{2}$ in the equation (2.3), we obtain:

$$D_0 = 2\overline{D_1} - L \times D_0\,, \tag{2.5}$$

or :

$$D_0 = \frac{2\overline{D_1}}{1+L}\,. \tag{2.6}$$

We assume that, from a certain level of partitioning process locality remains constant(or we ignore the variations by considering the mean) From this level of partitioning process we can write the following equations:

$$\begin{aligned} D_0 &= \frac{2\overline{D_1}}{1+L} \\ \overline{D_1} &= \frac{2\overline{D_2}}{1+L} \\ \overline{D_2} &= \frac{2\overline{D_3}}{1+L} \\ &\vdots \\ \overline{D_{K-1}} &= \frac{2\overline{D_K}}{1+L}\,. \end{aligned}$$

From previous equations, we deduct :

$$D_0 = \frac{2^K\overline{D_K}}{(1+L)^K}\,, \tag{2.7}$$

or, by taking the logarithm with base 2 of this expression :

$$\log D_0 = \log 2^K + \log \overline{D_K} - K\log(1+L)\,, \tag{2.8}$$

i.e.

$$\log D_0 = K(1 - \log(1+L)) + \log \overline{D_K}\,. \tag{2.9}$$

Since after the last level, after K bi-partitions, all partitions are indivisible, we have as many partitions as vertices in the hypergraph. We have then $K = \log B$, where B i ste total number of vertices. And we denote $\bar{t}$, as the mean degree of the vertices. We can then write the equation (2.9) in the form of Rent's Rule:

$$\log D = p\log B + \log \bar{t}\,, \tag{2.10}$$

with Rent exponent $p = 1 - \log(1+L)$.

We can notice that :

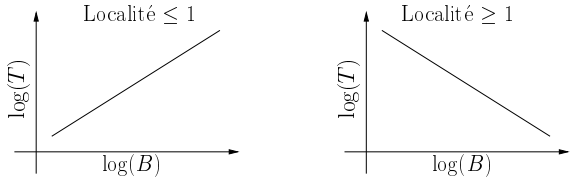


Figure 2.13: Effect of Locality on the Rent Plot.

- $p \geq 0$ pour $L \leq 1$ et
- $p \leq 0$ pour $L \geq 1$.

Or in other words :

- In the Zonz 2 of the Rent plot(cf. figure 2.11), the mean Locality of the sub-partitions is greater than 1. That is the terminals inside a partition are strongly connected among themeselves, and there are a few connections between the different partitions.
- However, in zone 1 (cf. figure 2.11), the Locality is small (less than 1), that is there are *in average* as much connections between different partitions as among the terminals inside the same partition(cf. figure 2.13).

The Rent's Rule helps us understand better, the place/Route constraints, hence it can used to optimise the concerned tools. For example, in the zone 2 of the Rent plot, the place/route will be easy since the Locality is very high(greater than 1), That is there are very few connctions between the blocks at this level. Whereas in the zone 1, place/route will be more constrained. The place/route tools thus could employ more effort in zone 1 than in zone 2, to obtain a good result.

2.4.1.3 Mutual Neighbours and feedthrough

Since the place/route of logic cell are done in a plane, we will take a closer look at the properties of the plane. We define that two partitions in a plane cna be mutual neighbours if and only if they have at least one edge in common. And n partitions are said to be Mutual Neighbours if each partition is neighbour to other $n-1$ partitions.

According to the Four-Color Theorem [71], we can't have more than four mutual neighbours in a plane. Hence if a netlis have partitions where more than four sub-partitions are connected among themselves(Frequently found in modern logic circuits), certain nets have to traverse another sub-partition(noy connected to it) to achieve its objective.

Because of this phenomenon, the average number of terminals per block(or partition) increases, and each net traversing a parttion without connecting to a cell (*i.e. feedthru*), in that partition, adds two more terminals to the partition it is traversing (cf. figure 2.14).

The constraints related to the Four-Color Theorem are not taken into account during the Rent Parameter caculations before place/route. Which implies the introduction of *Intrinsic Rent Parameters* and *Post-Placement Rent Parameters* [86, 65]. The intrinsic Rent Exponent and coefficient are the values computed through recursive partitioning, whereas Post-Placement Rent Exponent and coefficient are the values calculated after place/route. The Intrinsic Rent parameters thus define a minimum bound, and Post-Placement Rent parameters are a measure of quality for Place/Route tools.

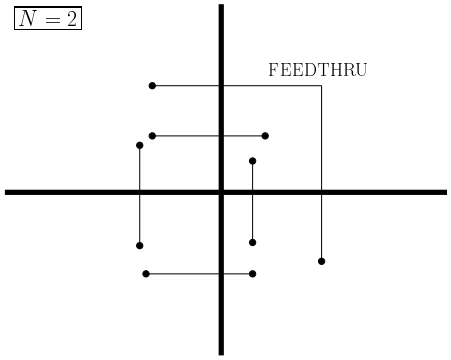


Figure 2.14: Mutual neighbours and Feedthrough.

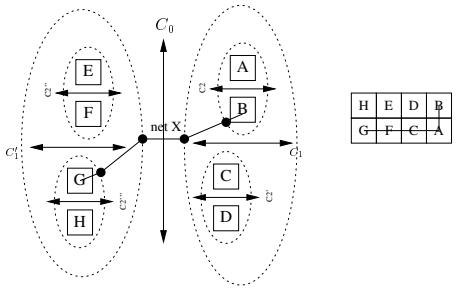


Figure 2.15: Influence of plane geometry on Post-Placement Rent Coefficient.

In the left of the figure 2.15, the partitioning method proposed previously allows us to detremine the intrinsic Rent parameters, since it does not take into account the constraints present in a plane. In the right, the routing between the cells are constrained by the plane geometry. The black wire is thus bound to traverse several partitions. Hence the Rent Parameters computed from the layout are worse. These are Post-Placement Rent Parameters.

2.4.1.4 Simulated Annealing

The simulated Annealing alogirithm [72, 64] allows us to obtain a good solution to NP-Difficult problem (often an extremum, *i.e.* a maximum or minimum), by random exploration and finally converging to a global extremum. Often used for placement of gates in ASIC/FPGA, simulated annealing tends to minimize a cost fucntion F equal to the sum of lengths of all the nets in a netlist.

In the case of FPGA, given the regularity of elementary cells, we have a proportional relationship between the length of a net and number of cells traversed. Hence we can write the cost fucntion F as:

$$F \doteq \sum_{i=1}^{N} Cn_i \,, \tag{2.11}$$

where Cn_i is the number of cell borders traversed by net i and N is the number of nets in the netlist. After recursively partitioning our netlist upto the elementary cells, we have done K bipartitions. Hence we have the following equalities:

- $2^K = B$ is the number of elementary cells.

- $\overline{D_K} = \bar{t}$, the mean degree of partitions is the mean degree of elemntary cells.

From (2.4), we can write:

$$B\bar{t} - D_0 = 2\left(\overline{C_1} + 2\overline{C_2} + 4\overline{C_3} \ldots + 2^{K-1}\overline{C_{K-1}}\right) .$$

In te case of placement of cells in a plane, it is necessary to consider the feedthrogh, which has the effect of increasing the mean degree of the partitions. We note in this case C_{ipp} and $\bar{t}_{pp}$ corresponds respectively to the value of the post-placement MinCut, and the Post-Placement mean degree of elementary cells.

$$B\bar{t}_{pp} - D_0 = 2\left(\overline{C_{1pp}} + 2\overline{C_{2pp}} + \ldots + 2^{K-1}\overline{C_{K-1pp}}\right) . \tag{2.12}$$

The Right hand side of the equation (2.12) is the sum of cutsizes of the netlist, which is equal to the number of cell borders traversed by all the nets in the netlist. From equations (2.12) and (2.11), we can write :

$$B\bar{t}_{pp} - D_0 = F = 2\sum_{i=1}^{N} Cn_i \, .$$

This equality tells us that by minimizing the total net length, we minimize the Post-Placement Rent coefficients. The intrinsic Rent cofficients being a minimum bound, we can use the difference between the intrinsic and Post-Placement coefficients as a quality measure of the place/route solution.

2.4.1.5 Application

Given a netlist, the computation of its Rent parameters allows us to have an estimation(minimum bound) of the surface necessary to place/route [61].

In the case of FPGAs, the Rents's Rule is highly pertinent, since the routing resources limit the size of netlists that we can realize in an FPGA(the routing resources constitute 80% of the surface of an FPGA [60]). The place/route in an FPGA can be considered as a projection of the netlist graph onto the graph of the FPGA. For a partition of the netlist to be routable on a FPGA partition, the following conditions are necessary:

- $B_{FPGA} \geq B_{netlist}$
- $T_{FPGA} \geq T_{netlist} \forall$ all concerned partitions

For each partition, we can show the Rent plot for a homogeneous Mesh FPGA(with a elementary cell 4 $\rightarrow$ 1 LUT + FF) is of the form $WB^{0.5}$ [86, 61], where W is the number of routing tracks per row and column. From this information, the conditions for routability of a given netlist are detremined as in figure 2.16.

The applications of the Rent's rule is becoming more and more relevant, given the evolution of the transistor dimensions, which is more and more close to that of routing wires. In future, even the ASICS could become constraind by the interconnect than the number of transistors.

In FPGAs, the routing resources takes up most of the surface area. In this case, Rent's Rule is a very helpful statistical tool which allows us to evaluate rapidly the routability of a given netlist in a FPGA.

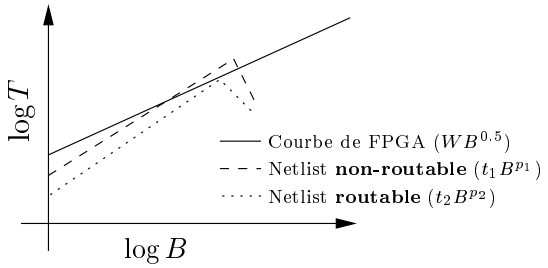
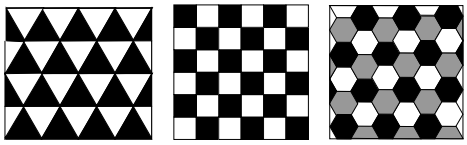


Figure 2.16: Routability of a given netlist in a simple mesh FPGA.



(a) 3.3.3.3.3.3 (b) 4.4.4.4 (c) 6.6.6

Figure 2.17: Regular Tiling Patterns.

2.4.2 Evaluation of Routing Architectures

The problem of placement and routing of gates in a plane can be best summarized with the four color theorem [71]. It states that every planar map is four-colorable, or in other words there can not be more than four mutual neighbours in a plane. For example in a square tiling pattern (exhaustively used in existing gate arrays) each tile has four neighbours but only two tiles can be mutual neighbours (or mutually adjacent). That is why a square tiling pattern is two-colourable. Note that two tiles are mutually adjacent when they share an edge. If in an imaginary tiling pattern all the blocks of a circuit could be mutual neighbours, each net will reduce to a unit length net. However in a square tiling if a net has a fan-out of more than 4, or more than 2 blocks are connected two each other, the concerned nets must traverse another block and thus use its routing channels. The advantage of square tiling pattern is that it can be easily laid out in silicon with only two interconnect layers. However present day semi-conductor technology offers us more flexibility than that, and that is why we are going to investigate the prospect of other tiling patterns. Several authors and notably Dehon [61] has pointed out the necessity to evolve the FPGA structures with technology (*i.e.* availability of more interconnection layers). Hierarchical gate arrays are one way of efficient utilization of available interconnect layers. In this manuscript we will consider array structures with different topologies.

Previous research efforts [69, 57] have investigated similar interconnect strategies for ASIC in detail, and this is even more relevant for FPGAs becuase of the reconfigurable switches associated with interconnects, which occupies the major portion of the total FPGA area.

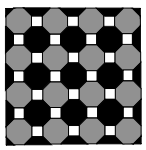
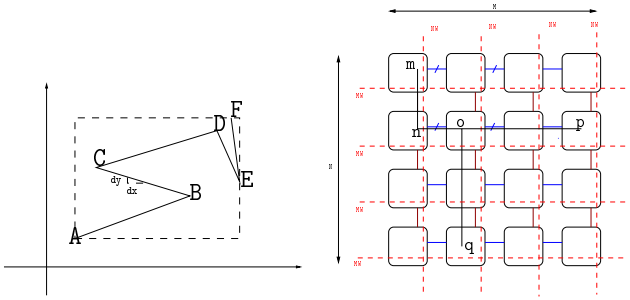


Figure 2.18: Octagonal (Truncated Square) Tiling Pattern (Semi-regular).



(a) Wire Flux (b) In a Mesh

Figure 2.19: Equivalence of wire flow and wire length.

The placement and routing of circuits depends precisely on two factors.

1. The characteristic of the netlist itself, which can be statistically expressed with the Rent Coefficient and Rent Exponent obtained by recursive bisection.

2. The characteristic of the tiling pattern, *i.e.* regularity, mutual adjacency, *etc.*

In this manuscript we will restrict ourselves to regular tiling patterns only, because their layout can be easily automatized. The rest of the paper is organized as follows. In subsection 2.4.3 we state the first principles on which this manuscript is based. In subsection 2.4.4 we analyze the hexagonal and octagonal tiling pattern based on Donath's analysis. In subsection 2.4.5 we compare the results obtained for these tiling patterns with square tiling. In subsection 2.4.6 we briefly discuss hierarchical gate array structures and we conclude.

2.4.3 First Principles

2.4.3.1 Tiling Patterns

Here we recapitulate the basics of tiling patterns formed with regular polygons. For a set of polygons forming a tiling pattern the internal angles of the polygons meeting at a vertex must add to 360°. Only three regular tiling patterns can be formed with regular polygons (figure 2.17). Any higher order polygon can not form a regular tiling pattern. For a regular tiling pattern all the vertices are indistinguishable from each other. However we will also be interested in a semi-regular tiling pattern formed by octagons known as truncated square tiling pattern, as shown in figure 2.18.

2.4.3.2 Equivalence of Wire Flow and Wire Length

Wire Flow: We define wire flow for a wire w.r.t. a boundary, as the magnitude of the dot product of the unit vector along the wire with the unit vector normal to that boundary. The definition of the flow ϕ of a wire w w.r.t. the boundary $x = X$ is

$$\phi_{|x=X} \doteq \left| \vec{U_w} \odot \vec{U_X} \right| . \tag{2.13}$$

Equivalence of Wire Length and Wire Flow: The length of a continuous monotonic and derivable curve from A to B can be expressed as(see fig 2.19(a))

$$\begin{aligned}
L_{wire} &= \int_A^B dl \\
L_{wire} &= \int_A^B \sqrt{dx^2 + dy^2} \\
L_{wire} &= \int_A^B \frac{dx^2 + dy^2}{\sqrt{dx^2 + dy^2}} \\
L_{wire} &= \int_A^B \frac{dx^2}{\sqrt{dx^2 + dy^2}} + \frac{dy^2}{\sqrt{dx^2 + dy^2}} \\
L_{wire} &= \int_A^B dx \cos\theta + dy \cos(90 - \theta) \\
L_{wire} &= \int_{X_A}^{X_B} dx \phi_{AB}(x) + \int_{Y_A}^{Y_B} dy \phi_{AB}(y) .
\end{aligned}$$

For a discontinuous/non-monotonic curve ABCDEF (see figure 2.19) the length can be calculated as the sum of the length of all continuous monotonic segments. We can express the above equation in terms of wire flow for a closed region between $(x_1, y_1), (x_2, y_2)$ with N_w monotonic continuous and derivable segments as

$$L_{wire} = \int_{x_1}^{x_2} \sum_{N_w} \phi(x) + \int_{y_1}^{y_2} \sum_{N_w} \phi(y) . \tag{2.14}$$

For example in fig 2.19(b) the wire `mnopq` has a length of 6 and $\phi_{mnopq} = 6$.

In terms of actual wirelength: The equivalence of wire length and wire flow is expressed as equation 2.14 when the actual distance is considered.

In terms of constituting segments or no. switches traversed: If the distance is calculated in terms of unit segment lengths (*i.e.* number of switches traversed), it can be stated as:

$$\text{Total Interconnect length} = \text{Nb of occupied segments/tile} \times \text{Nb of Tiles} .$$

We define a unit segment as a piece of wire between two switches in the array(diagonal, vertical or horizontal). The difference is shown in fig 2.20.

Square Tiling Pattern For a square tiling pattern the wire flow for each tile in the x and y direction is w(channel width). Hence for a square tiling pattern of size $M \times N$ and of channel width of w equation (2.14) can be simplified to:

$$\begin{aligned}
L_{wire} &= \sum_1^M \sum_1^N w + \sum_1^N \sum_1^M w \\
L_{wire} &= 2 \times B \times w ,
\end{aligned}$$

where B is the number of gates in the array. Thus for a given netlist with a total interconnect length L_{wire} the average channel width $\overline{w}$ in the Manhattan grid can be calculated to

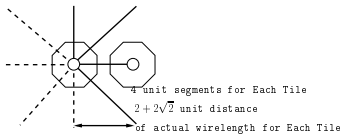


Figure 2.20: 4 segments per tile

be:

$$\overline{w} = \frac{L_{wire}}{2B}\,. \tag{2.15}$$

We note that for a uniform square tiling pattern the actual length and the number of switches traversed turn out to be the same.

Octagonal Tiling Pattern For an octagonal tiling pattern, the wire flow through a tile is(see fig. 2.22)

$$\phi_x = w + 2cos(45\,^\circ)w$$
$$\phi_y = w + 2cos(45\,^\circ)w$$

A octagonal pattern of size $M \times N$ and of channel width of w equation (2.14) can be simplified to:

$$\begin{aligned} L_{wire} &= \sum_{1}^{M}\sum_{1}^{N}[w + 2cos(45\,^\circ)w] + \sum_{1}^{N}\sum_{1}^{M}[w + 2cos(45\,^\circ)w] \\ L_{wire} &= (2 + 2\sqrt{2}) \times B \times w\,, \end{aligned}$$

where B is the number of gates in the array. Thus for a given netlist with a total interconnect length L_{wire}, the average channel width $\overline{w}$ in the Octagonal grid can be calculated to be:

$$\overline{w} = \frac{L_{wire}}{(2 + 2\sqrt{2})B}\,. \tag{2.16}$$

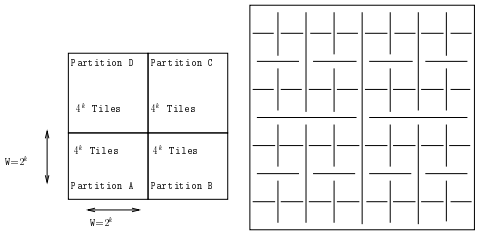
If we express the total interconnect length as the number of constituting segments (always integer) the expected channel width can be expressed as:

$$\overline{w} = \frac{L_{segments}}{4B}\,, \tag{2.17}$$

since there are 4 segments per tile(see fig 2.20). In a similar fashion the expected channel width for a hexagonal tiling pattern is

$$\overline{w} = \frac{L_{segments}}{3B}\,. \tag{2.18}$$

Although we discuss the equivalence in terms of actual wirelength and in terms of constituting unit segments(nb of switches traversed), hereafter we will be considering the distance in terms of constituting segments(number of switches traversed) only, since w.r.t today's technology the interconnect delay is negligible compared to delay in switches. However the other approach can be used if we calculate the physical wirelength(e.g in ASICS).



(a) Individual Partition (b) partitioning Process

Figure 2.21: Recursive Partitioning into squares à la Donath

This total interconnect length can be calculated accurately after the placement of the netlist or it can be estimated from the Rent Parameters of the netlist, which we will be discussing in subsection 2.4.4. Note that, this total interconnect length and average channel width is based on global routing only (*i.e.* all switchboxes are assumed to be full crossbars), which can change significantly depending on the depopulation scheme used in practice. Effect of depopulation schemes on channel width is out of scope of this manuscript since we want to compare tiling patterns independent of any depopulation scheme actually used.

2.4.3.3 Statistical Description of Netlists

Rent's parameters [75, 65, 58] (as explained in subsection 2.4.1) are statistical descriptions of a netlist which can be obtained by recursive bi-partitioning of a netlist.

Although Rent's Rule for certain netlists show different zones (*i.e.* p changes over different partition sizes) for our purpose we will consider that p is constant throughout the netlist partitions. Hereafter we will represent a user netlist with B gates as a 3-tuplet $\langle t, p, B \rangle$.

2.4.3.4 Donath's Analysis

Donath [62] has shown us an elegant way to calculate the average interconnect length of a netlist from its Rent parameters and the geometrical properties of the gate array. We briefly recapitulate the analysis made by Donath for reader's convenience [62]. The gate array is recursively partitioned into four equal squares as shown in figure 2.21(b). Let $\overline{r_k}$ be the average net length between squares of size 4^k at kth partition level and $\overline{n_k}$ be the average number of nets between squares of size 4^k at the kth partition level. Then the total interconnect length is given by:

$$L_{netlist} = \sum_{k=0}^{L-1} \overline{n_k} \times \overline{r_k} \,. \tag{2.19}$$

For a user netlist $\langle t, p, B \rangle$, the expected number of connections at each level of hierarchy is calculated to be:

$$\begin{aligned} \overline{n_k} &= N(4^k) - N(4^{k+1}) \\ \overline{n_k} &= \alpha t B (1 - 4^{p-1}) 4^{k(p-1)} \,, \end{aligned} \tag{2.20}$$

where $N(4^k)$ is the number of connections between groups of size 4^k which also includes connections between groups of size 4^{k+1}. Additionally, α is assumed to be $\frac{1}{2}$ considering only two terminal nets. This expected number of connections is same for square, octagonal or hexagonal arrays for a given user netlist $\langle t, p, B\rangle$, since this is a property of the netlist and we use the same partitioning process in each case. And for a Manhattan grid the average length between 4 square partitions is(with 4^k gates,see figure 2.21(a)) [62]

$$\overline{r_k} = \frac{4\overline{r_{adj}} + 2\overline{r_{opp}}}{6} \tag{2.21}$$

$$\overline{r_k} = \frac{14}{9}W - \frac{2}{9W} \tag{2.22}$$

$$\tag{2.23}$$

(where $W = 2^k$) since there can be four pairs of adjacent squares and two pairs of diagonally opposite squares in this partitioning scheme.

The total interconnect length is then calculated to be

$$\begin{aligned}\sum_{k=0}^{L-1} \overline{n_k} \times \overline{r_k} &= \sum_{k=0}^{L-1} \alpha t B(1-4^{p-1})4^{k(p-1)}\left[\frac{14}{9}W - \frac{2}{9W}\right] \\ &= \alpha t B(1-4^{p-1})\frac{2}{9}\left(7 \times \frac{B^{(p-\frac{1}{2})}-1}{4^{(p-\frac{1}{2})}-1} - \frac{1-B^{(p-\frac{3}{2})}}{1-4^{(p-\frac{3}{2})}}\right)\end{aligned} \tag{2.24}$$

Where $B = 4^L$

2.4.3.5 Comparison Method

Table 2.1: Notations

Symbol	Meaning
w	`Channel Width`
L_{wire}	`Total Interconnect Length in terms of physical wire length for a given netlist`
$L_{segments}, L_{netlist}, L_{tiling}$	`Total Interconnect Length in terms of unit constituting segments for a given netlist(in terms of hops)`
W	`Width of the kth partition of size` 4^K
t	`Rent Coefficient`
p	`Rent Exponent`
B	`Nb of blocks in a given netlist,` $B = 4^L$
L	`Nb of levels in the hierarchical partitioning process,` $B = 4^L$

After stating the first principles in the previous subsections, we now proceed to develop a method to compare tiling patterns.The Method is as follows

- First we calculate the total interconnect length L_{total} for a user netlist $\langle t, p, B\rangle$ according to Donath's method for each tiling pattern.

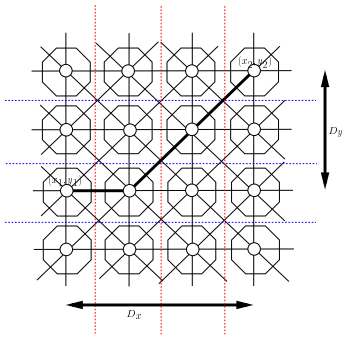


Figure 2.22: Distance Between two points in an Octagonal grid

- Next we calculate the average Channel width w_{av} required for each tiling patterns using the equivalence of wire length and wire flow.
- The no of switches required to implement a generic crossbar switchbox with w_{av} channels in each tiling pattern is obtained.
- Since most of the area of an FPGA is the area of its routing resources, we compare the efficiency of tiling patterns w.r.t the total no of switches in a switchbox required to implement a given user netlist $\langle t, p, B \rangle$

2.4.4 Analysis of Tiling Patterns

2.4.4.1 Truncated Square Tiling(Octagonal)

We will proceed to calculate the average length of nets between partitions of size $W = 2^K$ in a truncated square tiling pattern. The distance between two blocks(x_1, y_1) and (x_2, y_2) in an octagonal tiling pattern can be expressed as

$$\begin{aligned} r &= \sqrt{2}D_x + D_y - D_x \qquad D_y \geq D_x \\ r &= \sqrt{2}D_y + D_x - D_y \qquad D_x \geq D_y \end{aligned}$$

Where

$$D_x = |x_1 - x_2|, \qquad D_y = |y_1 - y_2|$$

The above relation can be expressed as

$$r = \frac{1}{\sqrt{2}} \left(D_x + D_y - |D_x - D_y| \right) + |D_x - D_y| \tag{2.25}$$

or

$$r = \frac{1}{\sqrt{2}} \left(D_x + D_y \right) + \left(1 - \frac{1}{\sqrt{2}} \right) |D_x - D_y|$$

However if we express the distance in unit segment lengths the distance between two points in a truncated square tiling pattern can be expressed as

$$\begin{aligned} r &= D_x + D_y - D_x \qquad D_y \geq D_x \\ r &= D_y + D_x - D_y \qquad D_x \geq D_y \end{aligned}$$

since we consider the diagonal to be of unit length. With respect to today's technology, the interconnect delay is negligible compared to delay in switches. Thus expressing the distance as the number of switches traversed is pertinent. The above equation can be expressed as

$$r = \frac{1}{2}\left(D_x + D_y + |D_x - D_y|\right) \tag{2.26}$$

The average distance between blocks in two adjacent square partitions is

$$r_{adj} = \frac{1}{W^4}\sum_{i_A=1}^{W}\sum_{j_A=1}^{W}\sum_{i_B=1}^{W}\sum_{j_B=1}^{W}\frac{1}{2}\left[(D_x + D_y) + |D_x - D_y|\right]$$

Where

$$D_x = |W + i_A - i_B|, \qquad D_y = |j_A - j_B|$$

$< i_A, j_A >$ and $< i_B, j_B >$ are two points belonging to partition `A` and partition `B` respectively and $W = 2^K$ as shown in figure 2.21(a).

The first term can be evaluated as follows

$$\begin{aligned}
r_{adj} &= \frac{1}{W^4}\sum_{i_A=1}^{W}\sum_{j_A=1}^{W}\sum_{i_B=1}^{W}\sum_{j_B=1}^{W} \\
&\quad \frac{1}{2}\left[(W + i_A - i_B + |j_A - j_B|) + |W + i_A - i_B - |j_A - j_B||\right] \\
r_{adj} &= \frac{1}{2}\left(\frac{4}{3}W - \frac{1}{3W}\right) + \frac{1}{2W^4} \times \\
&\quad \sum_{i_A=1}^{W}\sum_{j_A=1}^{W}\sum_{i_B=1}^{W}\sum_{j_B=1}^{W}\left[|W + (i_A - i_B) - |j_A - j_B||\right] \\
r_{adj} &= \frac{1}{\sqrt{2}}\left(\frac{4}{3}W - \frac{1}{3W}\right) + \frac{1}{2W^4} \times \times \\
&\quad \sum_{i_{AB}=-W}^{W}\sum_{j_{AB}=-W}^{W}\left(W - |i_{AB}|\right)\left(W - |j_{AB}|\right)|W + i_{AB} - |j_{AB}|| \\
r_{adj} &= \frac{1}{2}\left(\frac{4}{3}W - \frac{1}{3W}\right) + \frac{1}{2W^4} \times \frac{42W^5 + 10W^3 + 8W}{60} \\
r_{adj} &= 1.01W - 0.08W^{-1} + 0.06W^{-3}
\end{aligned}$$

similarly r_{opp} can be calculated to be

$$r_{opp} = \frac{1}{W^4}\sum_{i_A=1}^{W}\sum_{j_A=1}^{W}\sum_{i_C=1}^{W}\sum_{j_C=1}^{W}\frac{1}{2}\left[(D_x + D_y) + |D_x - D_y|\right]$$

Where

$$D_x = |W + i_A - i_C|, \qquad D_y = |W + j_A - j_C|$$

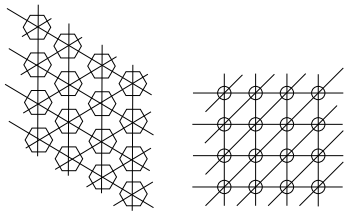


Figure 2.23: The Tile pattern exploded and transformed for analysis

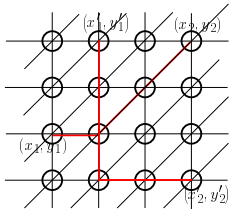


Figure 2.24: Distance in Hexagonal Tiling

The first term can be evaluated as follows

$$
\begin{aligned}
r_{opp} &= \frac{1}{2W^4}\sum_{i_A=1}^{W}\sum_{j_A=1}^{W}\sum_{i_C=1}^{W}\sum_{j_C=1}^{W}\left[(W+i_A-i_C)+(W+j_A-j_C)\right]+\frac{1}{2W^4}\times \\
&\quad \sum_{i_A=1}^{W}\sum_{j_A=1}^{W}\sum_{i_C=1}^{W}\sum_{j_C=1}^{W}\left|(i_A-i_C)-(j_A-j_C)\right| \\
r_{opp} &= W+\frac{1}{2W^4}\times \\
&\quad \sum_{i_{AC}=-W}^{W}\sum_{j_{AC}=-W}^{W}\left(W-|i_{AC}|\right)\left(W-|j_{AC}|\right)|i_{AC}-j_{AC}| \\
r_{opp} &= W+\frac{1}{2}\left[\frac{7}{15}W-\frac{1}{3W}-\frac{2}{15W^3}\right) \\
r_{opp} &= 1.23W-0.16W^{-1}-0.06W^{-3}
\end{aligned}
$$

Hence the average net length is calculated to be

$$r_{av} = \frac{4r_{adj}+2r_{opp}}{6} = 1.08W-\frac{0.073}{W}+\frac{0.02}{W^3} \tag{2.27}$$

2.4.4.2 Hexagonal Tiling

For the purpose of easy analysis we explode and transform the hexagonal tiling pattern as shown in 2.23. Although this transformation doesn't conserve the actual wirelengths, the wirelengths in terms of unit constituting segments is the same in both cases. This is done to use the same partitioning process for all tiling patterns. The distance between two points$(x_1, y_1)(x_2, y_2)$ on a hexagonal tiling in terms of unit segments(no of switches)

is given by(see fig 2.24)

$$\begin{aligned} r &= \frac{1}{2}\left(D_x + D_y + |D_x - D_y|\right) && (x_2 \geq x_1, y_2 \geq y1) \\ & && (x_2 \leq x_1, y_2 \leq y1) \\ &= D_x + D_y && otherwise \\ & \text{Where} \quad D_x = |x_1 - x_2|, \quad D_y = |y_1 - y_2| \end{aligned}$$

The above equation can be expressed as

$$\begin{aligned} 2r &= \left(1 + \frac{(x_2 - x_1)(y_2 - y_1)}{|x_2 - x_1|\,|y_2 - y_1|}\right)\frac{1}{2}\left[(D_x + D_y) + |D_x - D_y|\right] + \\ & \left(1 - \frac{(x_2 - x_1)(y_2 - y_1)}{|x_2 - x_1|\,|y_2 - y_1|}\right)[D_x + D_y] \end{aligned}$$

Now we proceed to calculate the average net length between two adjacent square partitions for a hexagonal tiling. Where

$$D_x = |W + i_A - i_B|, \qquad D_y = |j_A - j_B|$$

$$\begin{aligned} 2r_{adj} &= \sum_{i_A=1}^{W}\sum_{j_A=1}^{W}\sum_{i_B=1}^{W}\sum_{j_B=1}^{W}\left[\frac{3}{2}[D_x + D_y] + \frac{1}{2}|D_x - D_y|\right] \\ & - \frac{1}{2W^4}\sum_{iA=1}^{W}\sum_{iB=1}^{W}\sum_{jA=1}^{W}\sum_{jB=1}^{W}\left[\frac{(W + i_A - i_B)(j_A - j_B)}{|(W + i_A - i_B|\,|j_A - j_B|}\left[W + i_A - i_B + |j_A - j_B|\right]\right] \\ & + \frac{1}{2W^4}\sum_{iA=1}^{W}\sum_{iB=1}^{W}\sum_{jA=1}^{W}\sum_{jB=1}^{W}\left[\frac{(W + i_A - i_B)(j_A - j_B)}{|(W + i_A - i_B|\,|j_A - j_B|}\left|W + i_A - i_B - |j_A - j_B|\right|\right] \end{aligned}$$

$$
\begin{aligned}
2r_{adj} &= \frac{3}{2}[\frac{4}{3}W - \frac{1}{3W}] + \frac{1}{2}\left[\frac{42}{60}W + \frac{1}{6W} + \frac{8}{60W^3}\right] \\
&\quad - \frac{1}{2W^4}\sum_{i_{AB}=-W}^{W}\sum_{j_{AB}=-W}^{W}\left[\frac{(j_{AB})}{|j_{AB}|} \times [W + i_{AB} + |j_{AB}|]\,(W - |i_{AB}|)(W - |j_{AB}|)\right] \\
&\quad + \frac{1}{2W^4}\sum_{i_{AB}=-W}^{W}\sum_{j_{AB}=-W}^{W}\left[\frac{j_{AB}}{|j_{AB}|} \times |W + i_{AB} - |j_{AB}||\,(W - |i_{AB}|)(W - |j_{AB}|)\right] \\
2r_{adj} &= \frac{3}{2}[\frac{4}{3}W - \frac{1}{3W}] + \frac{1}{2}\left[\frac{42}{60}W + \frac{1}{6W} + \frac{8}{60W^3}\right] \\
&\quad - \frac{1}{2W^4}\sum_{i_{AB}=-W}^{W}\sum_{j_{AB}=0}^{W}[W + i_{AB} + |j_{AB}|]\,(W - |i_{AB}|)(W - |j_{AB}|) \\
&\quad + \frac{1}{2W^4}\sum_{i_{AB}=-W}^{W}\sum_{j_{AB}=-W}^{-1}[W + i_{AB} + |j_{AB}|]\,(W - |i_{AB}|)(W - |j_{AB}|) \\
&\quad + \frac{1}{2W^4}\sum_{i_{AB}=-W}^{W}\sum_{j_{AB}=0}^{W}|W + i_{AB} - |j_{AB}||\,(W - |i_{AB}|)(W - |j_{AB}|) \\
&\quad - \frac{1}{2W^4}\sum_{i_{AB}=-W}^{W}\sum_{j_{AB}=-W}^{-1}|W + i_{AB} - |j_{AB}||\,(W - |i_{AB}|)(W - |j_{AB}|)
\end{aligned}
$$

$$
\begin{aligned}
2r_{adj} &= \frac{3}{2}[\frac{4}{3}W - \frac{1}{3W}] + \frac{1}{2}\left[\frac{42}{60}W + \frac{1}{6W} + \frac{8}{60W^3}\right] \\
&\quad - \frac{1}{2W^4}\left(\frac{4W^5 + 3W^4 - W^3 - 4W^5 + 3W^4 + W^3}{6}\right) \\
&\quad + \frac{1}{2W^4}\left(\frac{21W^5 + 30W^4 + 5W^3 + 4W - 21W^5 + 30W^4 - 5W^3 - 4W}{60}\right) \\
r_{adj} &= 1.175W - 0.125W^{-1} + 0.03W^{-3}
\end{aligned}
$$

Now we calculate the average distance in a hexagonal tiling pattern between two opposite square partitions of width $W(2^k)$. We note that the average distance between the opposite squares A and C 2.21(a) is similar to the truncated square tiling and the average net length between the opposite squares B and D are similar to the Manhattan grid. hence r_{opp} is calculated to be

$$r_{opp} = \frac{1}{2}(W + 1.23W) \tag{2.28}$$

neglecting W^{-1} and W^{-3} terms. Hence The average distance in a hexagonal tiling pattern is

$$
\begin{aligned}
r_{av} &= \frac{4r_{adj} + 2r_{opp}}{6} \\
r_{av} &= 1.32W - 0.11W^{-1} + 0.01W^{-3}
\end{aligned} \tag{2.29}
$$

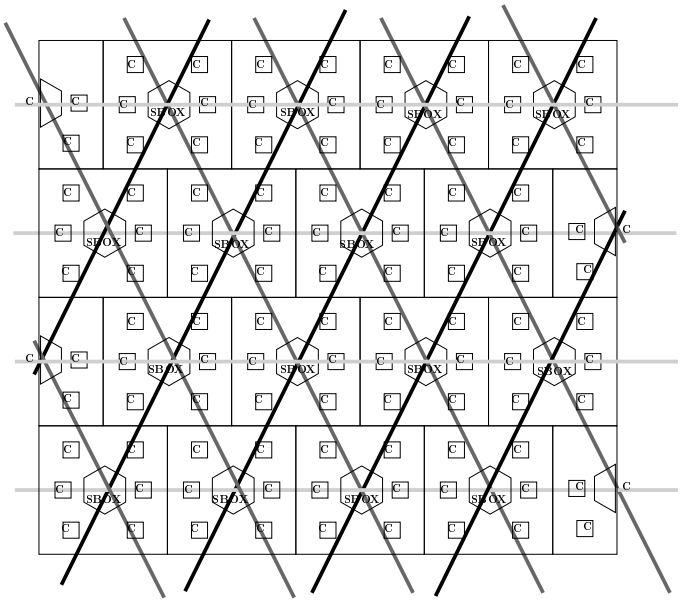


Figure 2.25: Hexagonal FPGA layout scheme

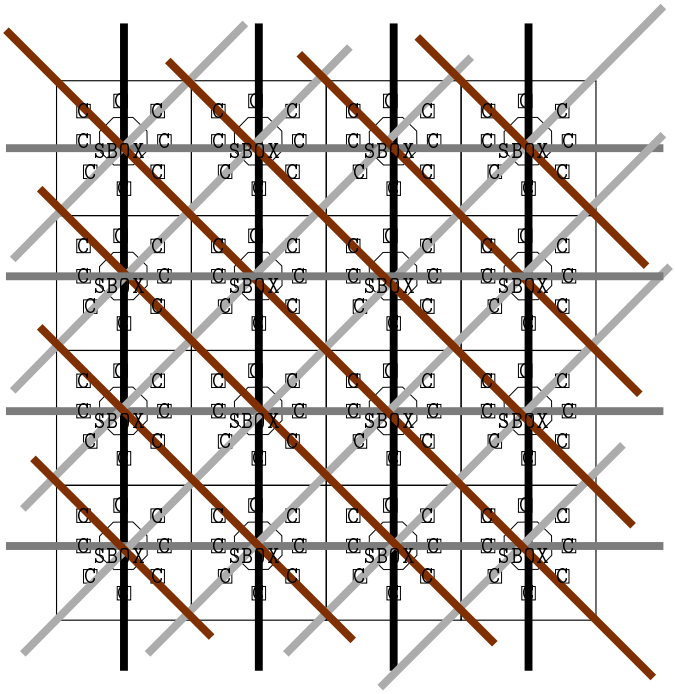


Figure 2.26: Octagonal FPGA layout scheme

2.4.5 comparison

2.4.5.1 Number of Switches

Let's calculate the number of switches required for a N-sided switchbox. For a crossbar with w tracks, between any two sides of the switchbox there are w^2 switches. And no of combinations of any two sides among N sides is C_2^N. Thus number of switches for the concerned tiling patterns are

$$\begin{aligned} S_{square} : S_{hexagonal} : S_{octagonal} &= \\ & C_2^4 {w_{square}}^2 : C_2^6 w_{hexagonal}^2 : C_2^8 w_{octagonal}^2 \\ S_{square} : S_{hexagonal} : S_{octagonal} &= \\ & 6w_{square}^2 : 15w_{hexagonal}^2 : 28w_{octagonal}^2 \end{aligned} \tag{2.30}$$

For disjoint(subset) switchboxes, there are W switches between any two sides of a Nèsided switchbox, Hence

$$S_{square} : S_{hexagonal} : S_{octagonal} = 6w_{square} : 15w_{hexagonal} : 28w_{octagonal} \tag{2.31}$$

2.4.5.2 Channel Width

The Total interconnect length for a user netlist $\langle t, p, B \rangle$ can be calculated from equation 2.19 and the average net length in each case(see eq 2.27, 2.29). We consider the three tiling patterns with equal number of blocks B, and since the partitioning process is the same in each case, the ratio of total interconnect length is(neglecting the W^{-1},W^{-3} terms)

$$L_{square} : L_{hexagonal} : L_{octagonal} = 1.55 : 1.32 : 1.08 \tag{2.32}$$

$$L_{square} : L_{hexagonal} : L_{octagonal} = 1 : 0.85 : 0.69 \tag{2.33}$$

The average no tracks w for each tiling pattern can be calculated to be

$$\overline{w_{square}} = \frac{L_{square}}{2B} \tag{2.34}$$

$$\overline{w_{hexagonal}} = \frac{L_{hexagonal}}{3B} \tag{2.35}$$

$$\overline{w_{octagonal}} = \frac{L_{octagonal}}{4B} \tag{2.36}$$

$$\overline{w_{square}} : \overline{w_{hexagonal}} : \overline{w_{octagonal}} = 0.78 : 0.44 : 0.27 \tag{2.37}$$

$$\overline{w_{square}} : \overline{w_{hexagonal}} : \overline{w_{octagonal}} = 1 : 0.56 : 0.35 \tag{2.38}$$

since total interconnect length in terms of unit segments must be equal to the number of occupied channel segments. Hence the total number of switches required to implement a

user netlist$\langle t, p, B\rangle$ is

$$\begin{array}{ccccc} S_{square} : & & S_{hexagonal} : & & S_{octagonal} \\ 6\left(\frac{L_{square}}{2B}\right)^2 : & & 15\left(\frac{L_{hexagonal}}{3B}\right)^2 : & & 28\left(\frac{L_{octagonal}}{4B}\right)^2 \\ 1 : & & 0.78 & & 0.57 \end{array}$$

Note that, this comparison is based on global routing where we assume that every switchbox is a crossbar. However in real FPGAs the switchbox is always depopulated and connection boxes are used. So we take a typical FPGA architecture with the following characteristics:

- disjoint switchbox (i.e no of switches in a switchbox is $C_2^N w$
- No of input/outputs to the CLB is T
- $w_{square} = w, \quad w_{hexa} = w \times 0.56, \quad w_{octa} = w \times 0.35$

Hence the total no of switches is given by

$$S_{square} = 6w + Tw \tag{2.39}$$
$$S_{hexagonal} = 15w \times 0.56 + Tw \times 0.56 \tag{2.40}$$
$$S_{octagonal} = 28w \times 0.35 + Tw \times 0.35 \tag{2.41}$$

e.g For T=16 the ratio for different tiling patterns is

$$\begin{array}{ccccc} S_{square} : & & S_{hexagonal} : & & S_{octagonal} \\ 1 : & & 0.78 & & 0.7 \end{array}$$

2.4.6 Hierarchical Gate Arrays

Although architectures used in practice always use an ad hoc depopulation scheme, for our analysis purpose we will consider a generic butterfly fat tree architecture as shown in figure 2.5. We denote arity A as the no of branches at each level, bifurcation ratio(α) as the ratio of channel width of a level to its next level, and w_k as the channel width at level k.

$$\alpha = \frac{w_{k+1}}{w_k} \tag{2.42}$$

For $A = 4$ the total no. of segments at level 0 is $B \times w_0$, at level 1 $\frac{B}{4} \times \alpha w_0$ and so on. Hence the total no of segments can be expressed as

$$\sum_{k=0}^{L-1} \frac{B}{4^k} \times \alpha^k w_0 = B w_0 \frac{\left(\frac{\alpha}{4}\right)^{(L)} - 1}{\frac{\alpha}{4} - 1} \tag{2.43}$$

If the total interconnect length for a netlist $\langle t, p, B\rangle$ is expressed as $L_{netlist}$, the expected value of w_0 can be expressed as

$$E(w_0) = \frac{L_{netlist}}{B\frac{\left(\frac{\alpha}{A}\right)^{(L)}-1}{\frac{\alpha}{A}-1}} \tag{2.44}$$

Now we proceed to calculate the total interconnect length according to equation 2.19. The distance(in terms of no of segments/switches traversed) between two points (x_A, y_A)

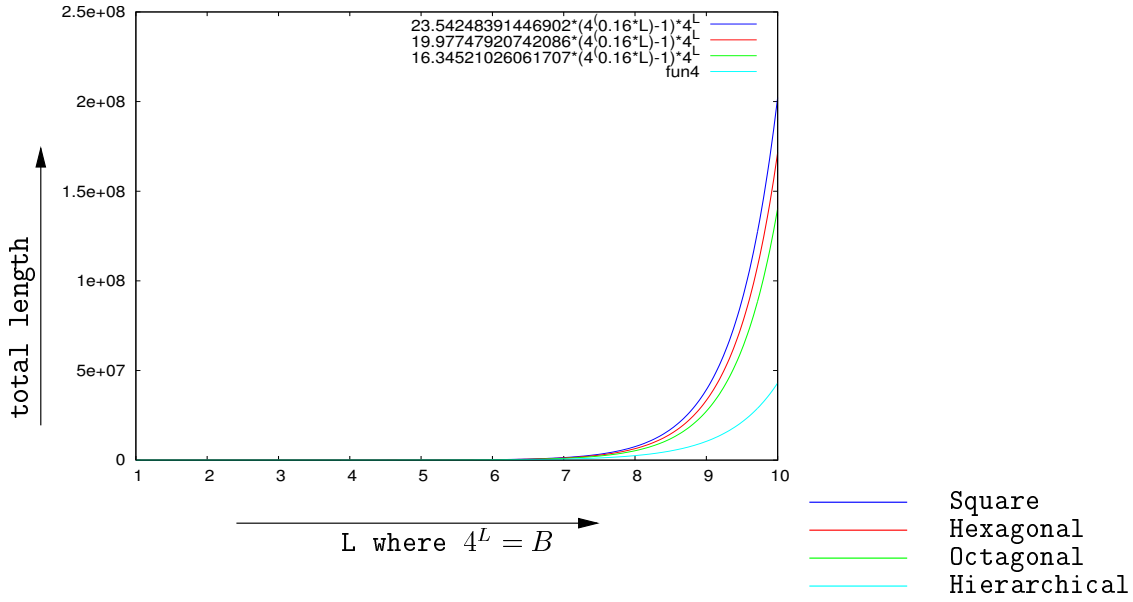


Figure 2.27: Total Interconnect length for different Tiling Patterns for a given user netlist $\langle 4, 0.66, 4^L \rangle$

and (x_B, y_B) belonging to two adjacent partitions of size 4^k is given by $r = (2k+1)$ The average length between two adjacent or diagonally opposite partitions is given by

$$\begin{aligned} \overline{r_k} &= \frac{1}{W^4} \sum_{i_A=1}^{W} \sum_{j_A=1}^{W} \sum_{i_B=1}^{W} \sum_{j_B=1}^{W} 2k+1 \\ \overline{r_k} &= 2k+1 \end{aligned} \tag{2.45}$$

where $W = 2^k$ as in fig 2.21(a)

The total interconnect length can be calculated using equation2.19 since we use the same partitioning process

$$\begin{aligned} \sum_{k=0}^{L-1} \overline{n_k} \times \overline{r_k} &= \sum_{k=0}^{L-1} \alpha t B (1 - 4^{p-1}) 4^{k(p-1)} \left[2k+1\right] \\ &= \alpha t B (1 - 4^{p-1}) \times \\ &\quad \left[\frac{(1 - 4^{(p-1)L})}{1 - 4^{p-1}} + \frac{2 \times 4^{p-1} \left[1 - L4^{(p-1)(L-1)} + (L-1)4^{(p-1)L}\right]}{(1 - 4^{p-1})^2} \right] \end{aligned} \tag{2.46}$$

In figure 2.27 we plot the total interconnect length for various tiling patterns for a given netlist$\langle 4, 0.66, 4^L \rangle$ w.r.t varying L. Note that we have neglected W^{-1}, W^{-3} terms in equation 2.24

This manuscript does a purely mathematical analysis based on generic models, and the idea is to investigate the possibility of using tiling patterns other than Manhattan grid in FPGAs. As shown in the previous subsections, use of tiling patterns formed with higher order polygons can improve the speed and area performance of an FPGA. These gain is highly dependent on depopulation schemes and other parameters. However for generic

tiling patterns with crossbar switchboxes there is a 22% gain in area for the hexagonal tiling pattern and a 30% gain in area for the octagonal tiling pattern. Moreover the average interconnect length is around 15% lesser for hexagonal and 31% lesser for the octagonal tiling compared to square tiling. We can expect a proportional increase in speed.

The physical realizability of these tiling patterns in CMOS are to be investigated. We show in figures 2.25 and 2.26, the layout schemes for hexagonal and octagonal FPGAs. `C` denotes the position of connection boxes inside the logic block and `SBOX` denotes the N-sided switch box. To our knowledge standard processes support 45° metal lines, whereas 60° lines can be etched using non-standard processes. We must keep in mind, that in practice one must use some sort of depopulation and staggering scheme, and these results provide only an idea of gains that can be achieved. The actual interconnect structure is of course dependent on several factors (i.e available interconnect layers, difficulty of fabrication, required speed/area, evolution of CMOS technology etc).

Chapter 3

Asynchronous Circuits

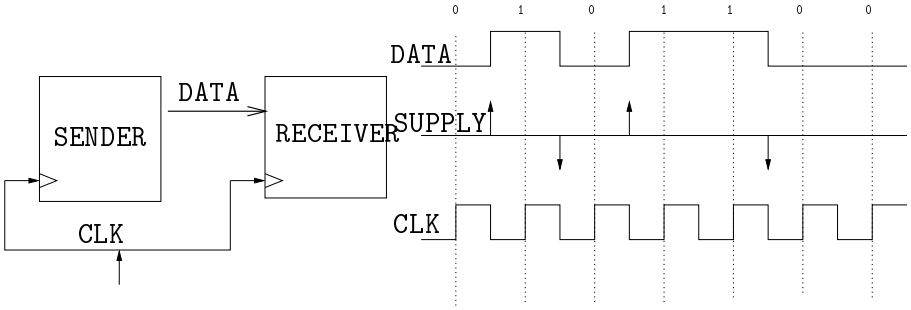
In this section we will discuss the key ideas of asynchronous logic. The author welcomes the reader to take a look in following publications for more detailed discussions [105, 93, 112]. Figure 3.1(b) shows the basic asynchronous handshake protocol. The sender puts valid data on the data line and sends a request on the REQ line to show the validity of data. Once the receiver has read the data, it assert the ACK line so that sender can put some new data. This basic protocol is formally defined as production rules or Seitz's weak conditions [111].

However this basic scheme is not hazard free. For example, if the data line has more delay than the REQ line. It's possible that receiver receives the request before the DATA is valid. To avoid such hazards the DATA and REQ are often encoded into a single channel. Most common encodings are 1-out-of-n encodings similar to one hot codes for state machines. Figure 3.1(c) depicts the 1-out-of-2 encoding. Table 3.1 describes the encoding scheme. Apart from encodings, the signaling protocol can also be of various flavors. The popular four phase protocol uses one phase for computation, and one phase to precharge all signals to zero state. Whereas 2-Phase protocols (NRZ) doesn't use precharge. Details about shows 2-phase Protocols can be found in [101].

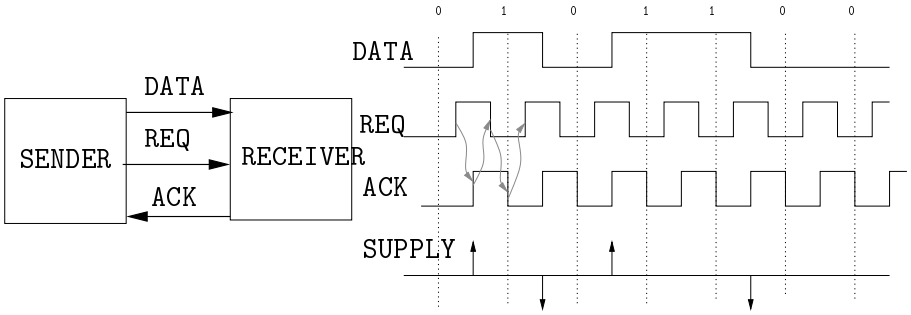
Table 3.1: 1-out-of-2, 4 Phase

Value	DATA0	DATA1
'0'	'1'	'0'
'1'	'0'	'1'
Precharge	'0'	'0'
Forbidden	'1'	'1'

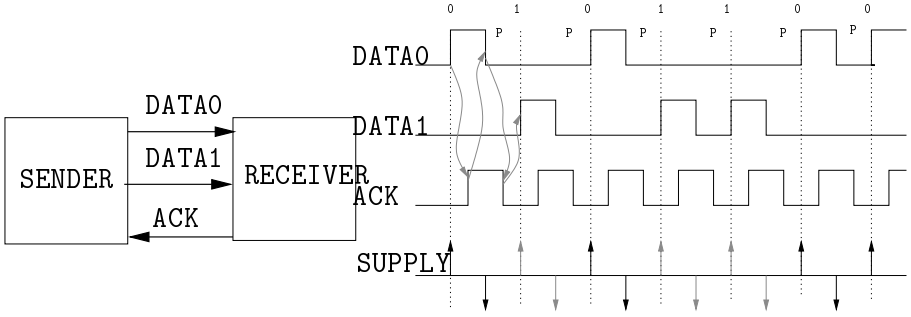
Since in a asynchronous signaling scheme each event has significance (as opposed to synchronous logic where a glitch can occur without disturbing the functionality), the asynchronous protocols are completely glitch free. Glitches are results of unbalanced input path arrival times(unbalanced joins). In Asynchronous circuits, this hazard is taken care of with C-Elements at the Gate level, however at the transistor level, there is a additional constraint of forks balanced in delays. This constraint is commonly known as "Isochronous Fork" constraint [105], and such asynchronous circuits are called Quasi-Delay Insensitive (QDI) Asynchronous circuits. In this manuscript we assume QDI asynchronous circuits implicitly whenever we discuss asynchronous protocols.



(a) Synchronous



(b) Basic Handshake Protocol



(c) 1-out-of-2 4-phase

Figure 3.1: Asynchronous Protocols

3.1 Timing Assumptions

This section provides a brief overview of timing assumptions behind asynchronous circuits. Note that this classification is neither exclusive nor exhaustive, and merely based on history of evolution of asynchronous circuit design. These assumptions are made by major lines of research over time, and often overlaps each other. Here we try to classify them in decreasing order of robustness.

Delay-Insensitive The functional correctness of this class of circuits is independent of any delays (finite non-negative) of constituting components and wires. This is by far the most general and most pessimistic assumption in terms of design(i.e we can see that the circuits should function over any range of temperature). This class contains most other classes defined hereafter.

Quasi-Delay-Insensitive(QDI) This class of circuits allow arbitrary(finite and non-negative) component and wire delays but any fork present in the circuit must be *Isochronous Fork*. An *Isochronous Fork* is such that the difference in delays between its fanout branches is negligible compared to the delay in the components to which it connects.

Speed-Independent This class of circuits make the assumption that there are no delays in interconnection wires, but components can have any non-negative and finite delays. We can see that if a wire is constrained to connect exactly two components, the subclass also satisfies delay-insensitivity.

Bounded-Delay Circuits This class of circuits assume lower and upper bounds on the various internal delays for correct and hazard-free behaviour. Modern Synchronous circuits fall into this category, the lower and upper bound being the *Setup* and *hold* time corresponding to a given clock frequency and clock skew.

μ-Pipeline The micro-pipeline approach, presented by Ivan Sutherland, uses single rail data bundles and separate wiring to indicate timing. So long as bundling constraints are met, a design centering on micro-pipelines is considered delay-insensitive.

Synchronous In synchronous circuits, the timing information comes from a global signal, and is the same for evere gate. Individual connections carry only data.

Other calsses of circuit frequently associated with asynchronous methodology are self timed circuits. This assumes that a circuit can be divided into "equi-potential" regions such that the delay in any wire within an equi-potential is assumed to be negligible, while delays in wires between two different "equi-potential" regions are assumed to be arbitrary. This class coincides with QDI if we assume that forks stay within an equi-potential region. Globally Asynchronous Locally Synchronous($GALS$) circuits are a mix of different styles. These circuits employ local clocks and closely follow synchronous design style for smaller modules that communicate asynchronously. This type of design is mainly motivated by the constraints of distributing a fast clock for very large designs.

3.2 Representation

3.2.1 Communicating Hardware Processes(*CHP*)

There exists a variety of semantics to represent asynchronous circuits, most of them are based on Communicating Sequential Processes [98]. The most popular is CHP, used as a standard language to describe asynchronous systems, and accepted by asynchronous synthesis tools(i.e SIS, TAST). There are several others such as HSE(*HandShake Expansions* PRs(*Production Rules*) to describe the design at various different levels of the design flow. They are often a subset of CHP, or a equivalent representation of CHP. For reader's convenience we reproduce verbatim a brief description of the semantics taken from [103] in table 3.2. A more elaborate description can be found in [95]

3.2.2 Petri Nets

The definition of a *Petri Net* is the following: (taken from Ref. [109]) A *Petri Net* is a 5-Tuple, $PN = (P, T, F, W, M_0)$ where:

- $P = p_1, p_2, \ldots, p_m$ is a finite set of places.
- $T = t_1, t_2, \ldots, t_n$ is a finite set of transitions.
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relation).
- $W : F \rightarrow 1, 2, 3, \ldots$ is a weight function.
- $M_0 : P \rightarrow 0, 1, 2, 3, \ldots$ is the initial marking.
- $P \cap T = \varnothing$ and $P \cup T \neq \varnothing$

A *Petri Net* $PN = (P, T, F, W)$ with initial marking M_0 is denoted as (N, M_0). A state or marking in a petri net is changed according to the following tarnstion(firing) rules:

- A transition t is said to be enabled if each input place p of t is marked with at least $w(p, t)$ tokens, where $w(p, t)$ is the weight of the arc from p to t.
- An enabled transition may fire.
- A firing of an enabled transition t removes $w(p, t)$ tokens from each input place p of t, and adds $w(t, p)$ tokens to each output place p of t.

A transition without any input place is called a *source transition*, and one without any output place is called a *sink transition*. A *source transition* is unconditionally enabled and a *sink transition* only consumes token without producing any. A *Petri Net* is said to be ordinary if all of its weights are '1'. A *Petri Net* is said to be pure if does not contain any self loops. A *self loop* occurs if p is both input and output place of transiton t.

Here we are going to discuss only major classes of Petri Nets and their properties, for a more detailed discussuion, the reader is directed towards Ref. [109].

Th emajor behavioural Properties of a *Petri Net* are

Table 3.2: Brief Description of CHP Semantics

Name	Notation	To be read as
Skip	skip	This statement does nothing.
Assignment	$x := E \quad , x\uparrow \quad , x\downarrow$	Assign the value of E to x. When E is *Boolean* we abbreviate $x\uparrow$ or $x\downarrow$
Communication	$X!e$	Send the value of e over channel X
Communication	$Y?x$	Receive a value over channel Y and store it in variable x. When we are not communicating data values over channel X, X denotes a synchronization action on port X
Probe	$\overline{X}$	The Boolean $\overline{X}$ is true *iff* a communication over channel X can complete without suspending.
Selection	$[G_1 \rightarrow S_1 [] \ldots [] \ldots G_n \rightarrow S_n]$	G_is are Boolean expressions(*guards*) and S_is are program parts. the execution of this command corresponds to waiting until one of the guards is true, and then executing one of the statements with a true guard. the notation $[G]$ is shorthand for $[G \rightarrow skip]$, and denotes waiting for the predicate G to become true. If the guards are not mutually exclusive we use the vertical bar $'\|'$ instead of $'[]'$.
Repetition	$*[G_1 \rightarrow S_1 [] \ldots [] \ldots G_n \rightarrow S_n]$	The execution of this command corresponds to choosing one of the true guards and executing the corresponding statement, repeating this until all guards evaluate to false. The notation $*[S]$ is a shorthand for $*[true \rightarrow S]$. If the guards are not mutually exclusive we use the vertical bar $'\|'$ instead of $'[]'$.
Sequential Composition	$S;T$	T is followed by S
Parallel Composition	$S\|\|T$	T and S are parallel actions
Simultaneous Composition	$S \bullet T$	The actions S and T complete simultaneously. Typically the two actions are communication actions only.

Reachability A marking M_n is said to be reachable from a marking M_0 if there exists a sequence of firings that transforms M_0 to M_n. The set of all possible markings reachable form M_0 in a net (N, M_0) is denoted by $R(N, M_0)$, and The set of all possible firing sequences from a marking M_0 in a net (N, M_0) is denoted by $L(N, M_0)$.

The *Reachbility* problem for a *Petri Net* is the problem of finding if $M_n \in R(N, M_0)$ for a given marking M_n in a net (N, M_0).

Boundedness (Safeness) A *Petri Net* (N, M_0) is said to be K-Bounded, if the number of tokens in each place doesn't exceed a finite number K for any marking reachable from M_0

$$M(p) \leq k \qquad \forall p \qquad \forall M \in R(M_0)$$

A *Petri Net* is safe if it is *1-bounded.*

Liveness A transition t in a *Petri Net* (N, M_0) is said to be:
L0-live or **dead** if t can never be fired in any firing sequences in $L(M_0)$.
L1-live if t can be fired at least once in some firing sequences.
L2-live if t can be fired at least k times in some firing sequences in $L(M_0)$, where k is any given integer.
L3-live if t can be fired infinitely often in some firing sequences in $L(M_0)$.
L4-live if t is **L1-live** for every marking M in $R(N, M_0)$

A *Petri Net* (N, M_0) is said to be Lk-live if every transition in the net is Lk-live. L4-liveness is comonly denoted as liveness.

We state here the major *Petri Net* subclassses of our interest and their corresponding *liveness* and *safeness* criteria for proofs see [109].

State Machine A *State Machine* is an ordinary *Petri Net* such that each transition t has exactly one input place and exactly one output place.

Theorem 3.2.1. *A State Machine* (N, M_0) *is live iff* N *is strongly connected and* M_0 *has at least one token.*

Theorem 3.2.2. *A State Machine* (N, M_0) *is safe iff* M_0 *has at most one token. A live State Machine* (N, M_0) *is safe iff* M_0 *has exactly one token.*

A Finite State Machine is a basic building block of both synchronous and asynchronous systems.

Marked Graph A *Marked Graph* is an ordinary *Petri Net* such that each place p has exactly one input transition and exactly one output transition.

Theorem 3.2.3. *A Marked Graph* (G, M_0) *is live iff* M_0 *places at least one token on each directed circuit in* G.

Theorem 3.2.4. *A live Marked Graph* (G, M_0) *is safe iff every arc(place) belongs to a directed circuit* C *with* $M_0(C) = 1$

A directed circuit is a closed sequence of places, tarnsitions and arcs where no arc is repeated. The Linder's synthesis algorithm(explained in subsection 3.6.1) uses the Marked Graph fromalism.

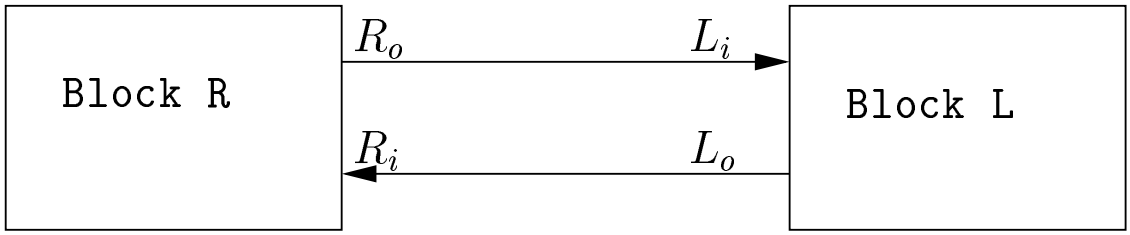


Figure 3.2: Handshake Protocols.

A Free-Choice Net A *Free-Choice Net* is an ordinary *Petri Net* such that every arc from a place is either a unique outgoing arc or a unique incoming arc to a transition.

Theorem 3.2.5. *A Free-Choice Net* (N, M_0) *is live iff if every siphon in* N *contains a marked trap.*

Theorem 3.2.6. *A live Free-Choice Net is safe iff* N *is covered by a strongly-connected SM(State Machine) components each of which has exactly one token at* M_0*.*

A siphon (deadlock) is a non-empty subset of places S in an ordinary net N, if every transition having an output place in S has an input place in S. A trap is is a non-empty subset of places Q in an ordinary net N, if every transition having an input place in Q has an output place in Q. The Free-Choice net formalism is used in Petrify synthesis method (explained in subsection 3.6.2) in an alternative form named STG(State Transition Graph).

State Transition Graph(*STG*) A *STG* is a equivalent representation of a Petri Net used in the Petrify open source tool. *STG*s are obtained from Petri Nets through following modifications:

- PN transitions are signal transitions.
- PN places and arcs are causal relations.
- Simple Places(One input arc, One output arc) are omitted.
- Place Markings are made on causal relation arrows.

3.3 Asynchronous Protocols

3.3.1 Two-Phase Handshake

The simplest handshake protocol between say right(R) and left(L) channel is called the two phase protocol. It is also called Non-Return to Zero(NRZ). The protocol is defined by the following handshake sequence in CHP notation:

Phase	Protocol	To be read as
R_{even}:	$R_o\uparrow; [R_i]$	set R_o to true; Wait for R_i to be true;
L_{even}:	$[L_i]; L_o\uparrow$	Wait for L_i to be true; Set L_o to true;

Since R_o and L_i are connected with the same wire as shown in figure 3.2, the only possible sequence of actions is $R_o\uparrow; L_i\uparrow; L_o\uparrow; R_i\uparrow$. But since all the handshake variables have become true after this phase we need a different handshake protocol for the next phase, which is

Table 3.3: N bit word represented by 1-out-of-K Code

K	No. of wires	No. of Transitions	Redundancy $1 - \frac{N}{2^{(\frac{N \times K}{log_2 K})}}$
2	$2 \times N$	N	$1 - \frac{N}{2^{2N}}$
3	$1.89 \times N$	$0.66 \times N$	$1 - \frac{N}{2^{1.89N}}$
4	$2 \times N$	$0.5 \times N$	$1 - \frac{N}{2^{2N}}$
5	$2.15 \times N$	$0.4 \times N$	$1 - \frac{N}{2^{2.15N}}$
6	$2.32 \times N$	$0.33 \times N$	$1 - \frac{N}{2^{2.32N}}$
7	$2.49 \times N$	$0.28 \times N$	$1 - \frac{N}{2^{2.49N}}$
8	$2.66 \times N$	$0.25 \times N$	$1 - \frac{N}{2^{2.66N}}$

$$\begin{array}{ll} R_{odd}: & R_o \downarrow; [\neg R_i] \\ L_{odd}: & [\neg L_i]; L_o \downarrow \end{array}$$

Thus for implementing two-phase handshake, we need to take into account these different behaviour depending on whether the block is in even or odd phase. Although two-phase handshake permits computing without any precharge phase, the circuit implementations are often complex.

3.3.2 Four-Phase Handshake

A solution to the above mentioned problem associated with even and odd phases is to reset all variables to reset(discharge) to their initial value $'0'$. This new protocol is known as *Four-Phase Handshake*(also known as *return-to-zero*). The protocol in CHP notation is:

$$\begin{array}{ll} R: & R_o \uparrow; [R_i]; R_o \downarrow; [\neg R_i] \\ L: & [L_i]; L_o \uparrow; [\neg L_i]; L_o \downarrow \end{array}$$

From figure 3.2 we can see that the possible set of actions is

$$R_o \uparrow; L_i \uparrow; L_o \uparrow; R_i \uparrow; R_o \downarrow; L_i \downarrow; L_o \downarrow; R_i \downarrow$$

This is same as the sequence described in figure 3.1(c).

3.3.3 DI Codes

As explained in the introduction of this chapter, *data* and *handshake* variables are often encoded together to avoid hazards. We also introduced 1-out-of-2 codes. This code belongs to a more general class of codes called 1-out-of-N codes. Although there could be a variety of other codes for handshake and data variables, in this section we will discuss only 1-out-of-N and LEDR codes.

1-out-of-N Codes 1-out-of-K Codes require K wires to represent $log_2 K$ bits of data, each wire representing a bit. Setting all wires to zero signifies the *reset* state. This is similar to one-hot encoding for state machine implementation, and the logic of coding is such that from one value to another, we only require a single transition. This is to avoid *race* conditions and *hazards*.

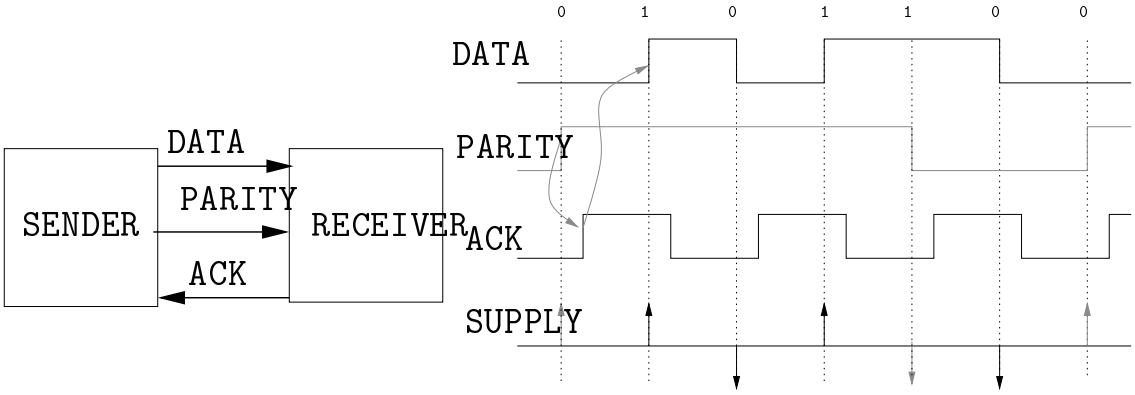


Figure 3.3: LEDR with two-phase Handshake.

We need $\frac{N \times K}{log_2 K}$ wires to represent a N-bit word with 1-out-of-K codes. Whereas, to set the wires to a valid value we need $2 * \frac{N}{K}$ transitions. [104]. Table 3.3 gives a comparative chart for number of wires and transitions for various values of N. We also note that compared to the case where each wire represents a bit of data, increasing K increases the amount of redundancy. We define *Redundancy* as the number of unused codes out of all possible codes. To represent N-bit word with 1-out-of-K codes we need $\frac{N \times K}{log_2 K}$ wires. Thus No. of possible codes are $2^{\frac{N \times K}{log_2 K}}$. Hence Redundancy is calculated as $1 - \frac{N}{2^{(\frac{N \times K}{log_2 K})}}$. We also show in table 3.3 this value of redundancy.

From table 3.3 we can see that 1-out-of-3 code is also a very interesting code, in terms of number of wires and transitions, while representing a N-bit word. Because of this reason, our asynchronous FPGA architecture is Multi-Style which will enable us to evaluate several codes and handshake protocols in terms of efficiency. However because of it's simple implementation, 1-out-of-2 dual-rail rail code is the most popular in asynchronous design.

LEDR(*Level Encoded Dual-Rail*) *Level Encoded Dual-Rail* is a coding associated with two-phase handshake, in which the *data* wire is same as the date being transmitted, and the *parity* wire toggles if there is a repetition of data. In figure 3.3 we can see that it's power consumption profile is not as symmetric as that of figure 3.1(c), and this also follows the scheme of only one transition for change of value. LEDR is particularly useful for one implementation of asynchronous FPGA commonly known as *Phased Logic*

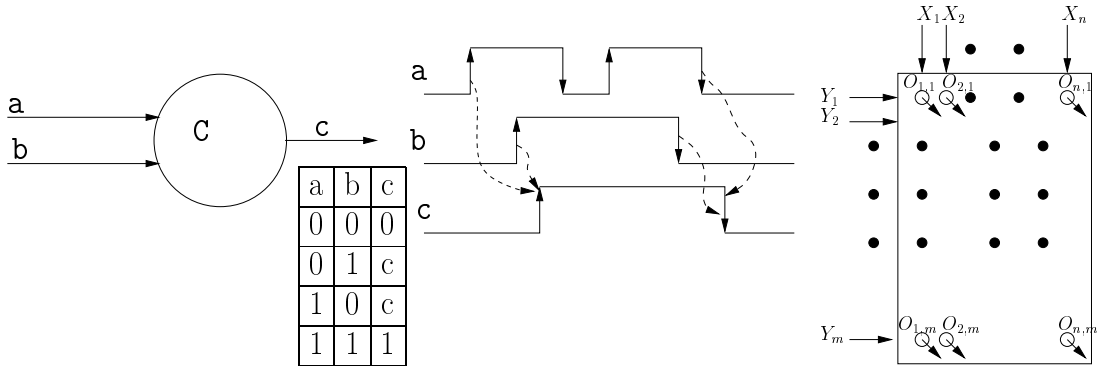
3.4 Building Blocks

3.4.1 C-ELEMENT

The Muller C-Element is the basic building block of asynchronous circuits, which synchronizes two independent events. In CHP notation C-Element can be described as

$$*[A \wedge B \rightarrow C \uparrow [] \quad]A \wedge]B \rightarrow C \downarrow]$$

This basic behaviour of a 2-input C-Element is shown in figure 3.4(a) with the corresponding truth table.



a	b	c
0	0	0
0	1	c
1	0	c
1	1	1

(a) Muller C-Element. (b) $m \times n$ Decision-Wait.

Figure 3.4: Synchronizing Elements.

From figure 3.4(a) we can see that a glitch on wire A is completely ignored by the C-Element output. This is the basic synchronizing action of a C-Element. i.e. the gate evaluates only when both of the events have occurred.

3.4.2 Decision Waits

The Decision Wait(DW) element is a generalization of the above mentioned Muller C-Element. The CHP specification of the DW element with n row inputs(r), and m column inputs(c) and corresponding $n \times m$ array of outputs is [100]

$$DW_{n\times m} = [_{1\le i\le n}|X_i? \rightarrow [_{1\le j\le m}|Y_j? \rightarrow O_{i,j}; DW_{n\times m}]]$$

The well-known Muller C-Element is a $DW_{1\times 1}$. A Decision-Wait waits for an event from both dimensions(events are mutually exclusive in a dimension) and sends an event on the output matching both indexes. If there are more than one transitions in one dimension, the output is undefined. Similar to C-Element, a DW is used to synchronize between two sets of events.

Sometimes when an input transition occurs on row wires, one may know that the environment will not send an event on some set of column inputs. In such case the implementation of the DW may be simplified. We call this new Decision-Wait a SDW(Sparse Decision-Wait). A SDW can be specified as

$$SDW_{n\times m} = [_{1\le i\le n}|X_i? \rightarrow [_{1\le j\le m}|s_{i,j}\&Y_j? \rightarrow O_{i,j}; SDW_{n\times m}]]$$

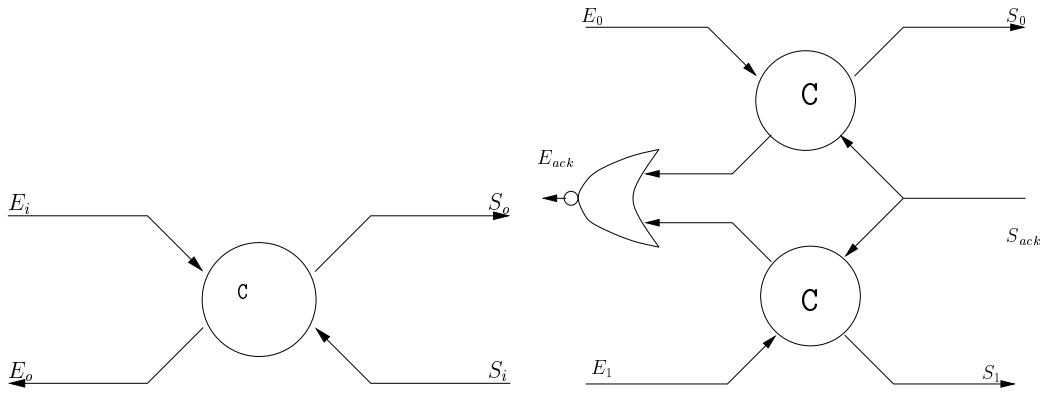
Figure 3.4(b) depicts this basic building block of asynchronous circuits.

3.5 Building Blocks w.r.t 1-out-of-2 4-Phase

3.5.1 Weak Condition Buffers(*WCHB, WCFB*)

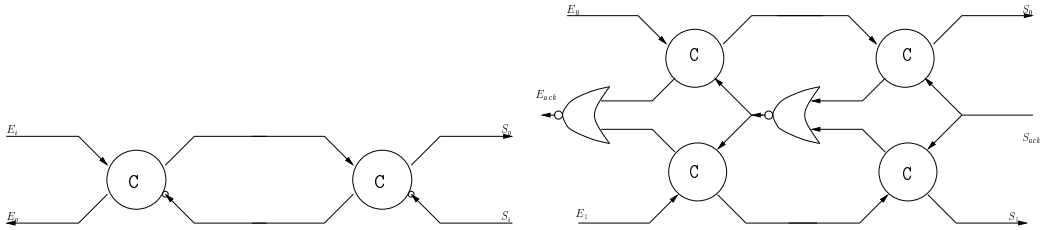
The basic sequencing building block in asynchronous design is known as *buffer*, which transmits data from the input environment to the output channel.

The simplest buffer, introduced by Muller [108] is known as *Weak-Condition Half Buffer*. Figure 3.5(a) depicts the handshaking signals for a WCHB with input channel



(a) Bare Handshake (b) With one bit of data

Figure 3.5: Half-Buffer



(a) Bare Handshake (b) With one bit of data

Figure 3.6: Full-Buffer

E and output channel S. The specification in CHP notation is:

$$*[[\lceil S_i]; [E_i]; E_o\uparrow; S_o\uparrow; [\lceil E_i]; [S_i]; E_o\downarrow; S_o\downarrow]$$

This has a very simple implementation as shown in figure 3.5(a).

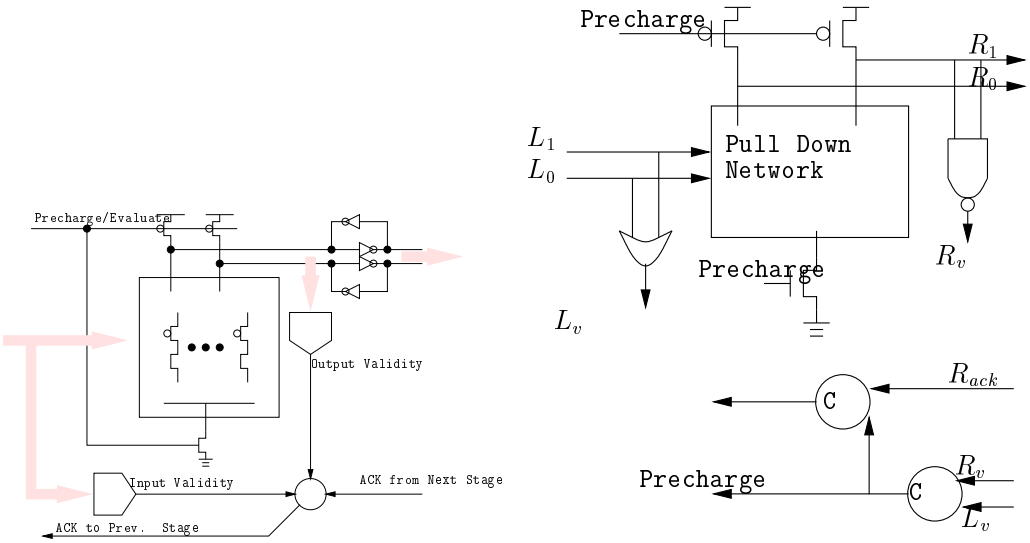
Another buffer which decouples the input and output handshaking sequence is called the *Full Buffer*. In terms of CHP specifications:

$$*[[E_i]; E_o\uparrow; [\lceil S_i]; S_o\uparrow; [\lceil E_i]; E_o\downarrow; [S_i]; S_o\downarrow]$$

From the CHP specifications we can see that the handshaking sequence of the input channel E can terminate without the sequence of output channel S being started. The names "full" and "half" comes from the fact that a full buffer is just two half buffers in series. Figure 3.5(b) 3.6(b) shows a half buffer and a full buffer with one bit of data with 1-out-of-2 Four-phase handshaking sequence.

3.5.2 Pre-Charge Buffers(*PCHB,PCFB*)

Pre-Charge buffers has the same functionality of buffers, with the advantage of high throughput. Figure 3.7(a) shows the PCHB implementation of a function, And figure 3.7(b) depicts the Pre-charge Half Buffer. Figure. 3.7(a) depicts the generic prcharge implementation of any function. The precharge signal is calulated from the input and output validity and the acknowledge from the next stage. The input and output validity trees are marked



(a) Generic Precharge Computation Circuits (b) Precharge Half-Buffer

Figure 3.7: Pre-Charge Computation

as blackboxes, since their actual implementation depends on the DI code used in the circuit. During the precharge phase the internal nets are precharged to '1'(output invalid). when input data is valid, ouput data is invalid, and "ack" from next stage is asserted, the gate evaluates by pulling down the outputs throgh a network of NMOS transistors. This netwok depends on the function to be computed. After evaluation, the "ack" to the previous stage is also asserted as soon as "ack" from the next stage is asserted. Now the gate waits for input data to become invalid, toggles to the precharge phase. PCHB computes the output, while the input port still contains the input data, thus eliminating the need of registered variable. Furthermore it completes the output handshake even before the input handshake. or the output channel is precharged. In CHP we can describe it as:

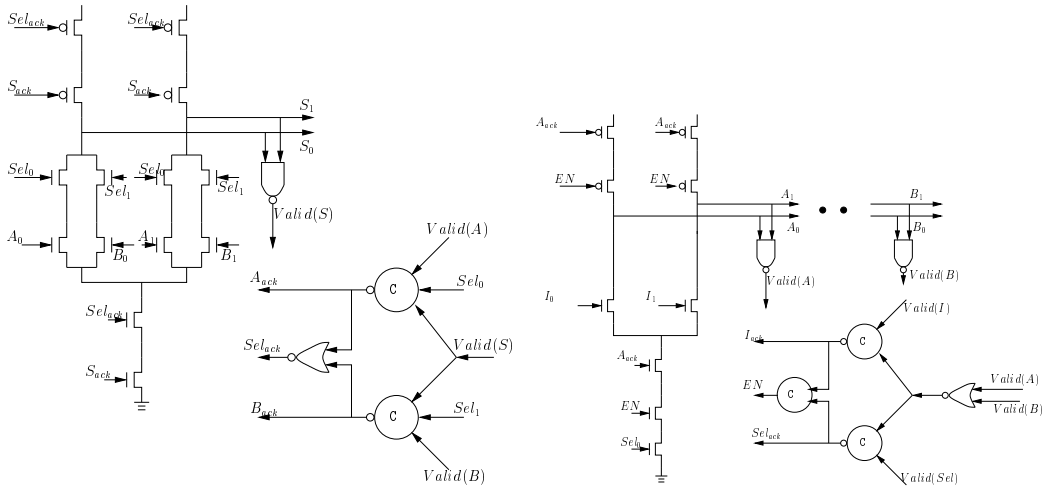
$$*[[R_{ack}]; [f0(L) \rightarrow R_0 \uparrow [] f1(L) \rightarrow R_1 \uparrow]; L_{ack} \downarrow; [\rceil R_{ack}]; R_0 \downarrow; R_1 \downarrow; [\rceil L_0 \wedge \rceil L_1]; L_{ack} \uparrow]$$

3.5.3 MERGE

A MERGE merges two input streams from input ports A and B into an output stream on port S. Which port to select for the next input communication is determined by the value received on control port *Sel*. In CHP notation:

$$*[Sel?sel; [Sel_0 \rightarrow S!(A?)[]Sel_1 \rightarrow S!(B?)]]$$

The implementation is shown in figure 3.8(a). The merge function is implemented using the generic function implementation described in figure. 3.5.2. The fucntion of selection and the two concerend inputs are implemented using the NMOS pull-down network, the input/output validity and the precharge signal compuatation is same as described in figure. 3.5.2



(a) Two-Way Controlled Merge. (b) Two-Way Controlled Split

Figure 3.8: Pre-Charge Implementation of Merge and Split

3.5.4 SPLIT

The SPLIT receives data on one input port S and sends it on one of two output ports A and B. Which output port is determined by the value received on control port *Sel*. In CHP notation:

$$*[Sel?sel; [Sel_0 \rightarrow A!(S?)[]Sel_1 \rightarrow B!(S?)]]$$

The implementation is shown in figure 3.8(b) The split function is implemented using the generic function implementation described in figure. 3.5.2. The fucntion of ouput selection and the input are implemented using the NMOS pull-down network, the input/outputs validity and the precharge signal compuatation is same as described in figure. 3.5.2

3.6 Synthesis Methods

3.6.1 Linder's Method

Linder [101] proposed a method to transform synchronous circuits to asynchronous circuits. This approach has the advantage, that the designer doesn't need to know about the intricacies of asynchronous design, and uses conventional synchronous design tools. The circuit is then automatically transformed to a self-timed implementation.

In the method proposed by Linder, all combinational gates are transformed to asynchronous token processing gates, Linder's algorithm adds the necessary acknowledgment signals to these gates. The Flip-Flops are transformed to what is known as *Barrier Gates*, which are the same as token processing gates, but generates the Tokens at startup. Here follows a pseudo-code of Linder's algorithm.

---

*/*Step One Building the Basic Topology */*
Copy the topology of the clocked system except **for** the clocked signal.
Each signal is replaced with its LEDR counterpart and each gate is replaced with its phased logic counterpart.

*/*Step Two: Guarantee Safeness */*

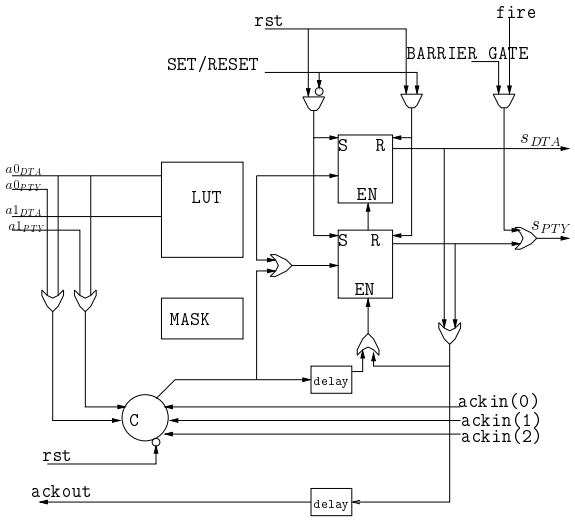


Figure 3.9: Phased Logic Cell for FPGA.

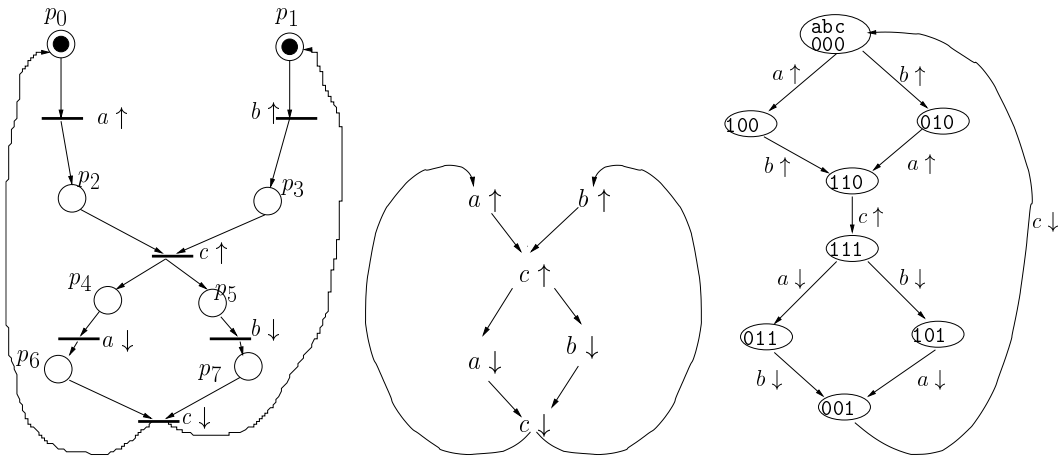
Add splitter Gates **if** necessary to separate barrier gates that are directly connected
/ Mark Safe Signals */*
for each barrier gate `$g_i$`
    Perform a Backward Depth−First Search starting at `$g_i$` that avoids barrier gates.
    Perform a Forward Depth−First Search starting at `$g_i$` that avoids barrier gates.
        Mark signals found in both searches as covered.
*/*Add Feedback to cover the remaining signals*/*
While there are unsafe signals
    for each gate g_i
        Perform a backward depth−first search starting at g_i along clear paths.
        for each gate g.found in the backward depth−first search
            Perform forward depth−first search along clear paths starting at
                g_j to determine which, **if** any, signals are covered **if** a feedback signal
                is added from g_i to g_j.
            Calculate the score **for** a feedback signal from g_i to g_j.
            If the score is the best seen so **far for** any g_i, save the feedback signal
            as the current best.
    Add the best feedback signal seen to the original phased logic network and mark all the
    signals that are covered by it.

In brief, Linder's algorithm adds the signals necessary for asynchronous operation, and then checks for the *Liveness* and *Safeness* criteria of the corresponding Petri Net. It adds more signals to make the Petri Net Live and Safe. The reader might note that, although the original Linder method was designed keeping in mind LEDR data encoding, and 2-phase handshake, this can be applied to other logic styles with slight modifications.

Figure. 3.9 shows the schematic of the elementary FPGA CLB that we used to evaluate Linder's approach. The signal `FIRE` denotes the start operation, where all tokens are generated. The signal `B-Gate` puts the cell in token generating mode.

3.6.2 Petrify

Unlike Linder's algorithm *Petrify* [93] is a pure-play asynchronous synthesis tool, developed by UPC, Spain with Jordi Cortadella as the main author. Petrify closely follows the popular open source synthesis tool *SIS*. The input to the petrify synthesis tool is a syntax very



(a) Petri Net Representation (b) State Transition Graph (c) State Assignment

C \ AB	00	01	10	11
0	0	0	0	1
1	0	1	1	1

$c_{next} = c(a\overline{b} + \overline{a}b)$

(d) Next State Equation

Figure 3.10: Petrify Synthesis

similar to *CHP*, which is then interpreted by *Petrify* as a Petri Net. This Petri Net is then transformed to a State Transition Graph.

In the next step, state values are assigned to this *STG* and next state equations are calculated. The interested reader might see reference. [93] for more details.

In Fig. 3.10 we describe this synthesis method with a simple example of C-Element synthesis. There are three sections, first to declare the inputs and outputs(i.e signals which are visible form the environment), next section describes the petri net, and the last section describes the initial markings.syntax a+ corresponds to the $a\uparrow$ transition in figure 3.10(a). Note that it is not necessary to explicitly specify places with only one input transition, and one output transition.

```
.model celement
.inputs a b
.ouputs c

.graph
a+ p2
b+ p3
p2 c+
p3 c+
c+ p4 p5
p4 a-
p5 b-
a- p6
b- p7
p6 c-
```

```
p7 c-
c- p0 p1

.marking{p0,p1}

.end
```

Fig. 3.10(a) shows the Petri Net representation of a Muller C-Element, which has been translated by *Petrify*. This Petri net is checked for Liveness and Safeness, and then it is transformed to it's corresponding *STG* shown in 3.10(b). In Figure. 3.10(c) the state values are assigned. Note that the state values are assigned in such a way that there is only one signal transition for going from one state to another, to minimize hazards and race conditions. In Figure. 3.10(c) this transition is same as the changing state variable. Figure. 3.10(d) shows the karnaugh map of the next state equations, where we see the same truth table of Muller C-Element. Petrify has additional capabilities of timing verification, and mapping the circuit to a given given library.The outputs are in *.blif* format as in *SIS*.

Chapter 4

State of the Art and Motivation

4.1 Successful Side-Channel Attacks and Counter-Measures

Side-Channel Attacks have evolved since its first publication by Paul Kocher [23]. Here we provide the results of recent attacks on a unprotected hardware DES implementation. These results are taken from the "DPA Contest" [116]. We can see that cipher can be broken with as small as 135 traces using some mathematical techniques. This shows the vulnerability of cryptographic circuits for such attacks. We don't have the data on some recent attacking method such as Template Attacks. The author invites the reader to keep an eye on the "DPA Contest" [116] website for latest results.

In table 4.2 we list some recent results of succsesful counter-measures against side-channel attacks. We mention their MTDs (Measurement to Disclosure) required to attack the hardware with and without the counter-measure, to give the reader an idea about the kind of gains these counter-measures can achieve. The reader may note that the counter-measures only increase the number of traces required to break the cipher. A full-proof counter-measure has not been reported yet in the literature.

Table 4.1: DPA Contest Results 2008

Author	Attacking Method	Traces
Yongdae KIM, Tohoku university Aoki laboratory	CPA using Pearson's correlation	135
Eloi SANFELIX, Riscure	targets 2 sboxes at a time (12 key bits), using a sum of the correlation above a certain threshold as a criterion	232
Victor LOMNE, LIRMM	enhancement of the CPA (from the work of Thanh-Ha LE et al.) on the 16th round of the DES selecting the good temporal window	310
Benedikt GIERLICHS, KUL	Difference of Means on the last round	329
Sylvain GUILLEY, ENST	multi-bit CPA with fourth-order cumulant preprocessing	569

Table 4.2: Effective Attacks and Counter-Measures

Reference	Standard	Counter-Measure	MTD	
			Unprotected	Protected
[25]	DES	Masking	10,000	200,000
	AES		320	21,185
[2]	AES		25,000	30,000
[33]	AES		25,000	130,000
[38]	CPU		279	471
[142]	DES	WDDL	18,304	Greater than 6,400,000
[142]	DES	seclib+backend duplication	18,304	Greater than 6,400,000

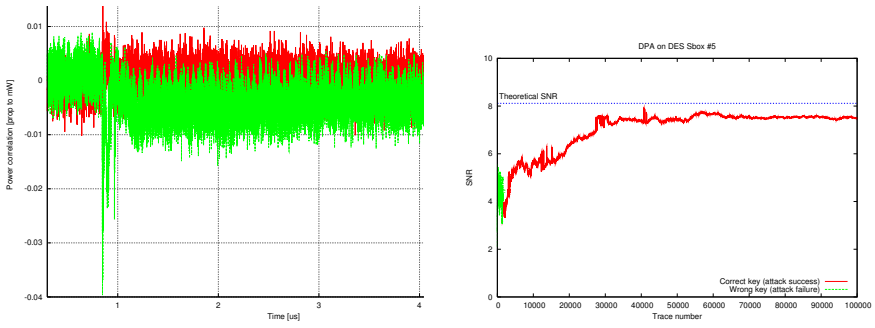


Figure 4.1: DPA on DES (in an Altera EP1S25B672C7) first round: Bad and correct subkey (*left*) and signal-to-noise ratio of the attack (*right.*)

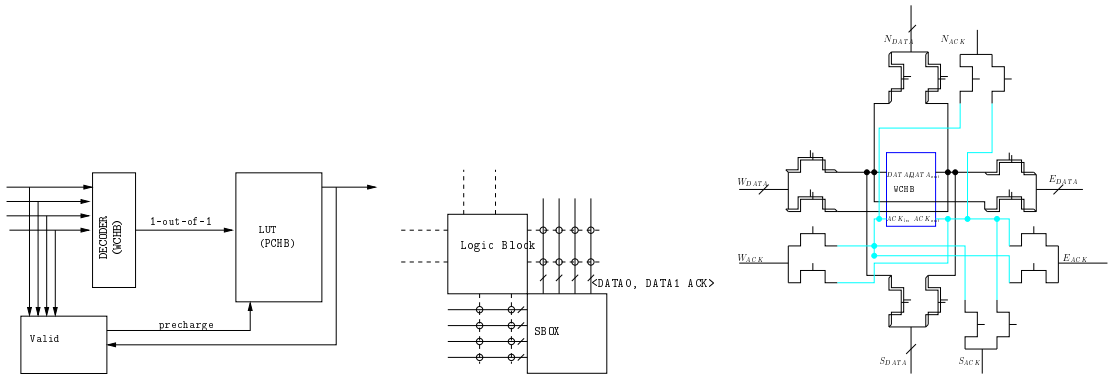
4.1.1 Side-Channel Attacks on FPGAs

Although the first SCA has been realized on a commercial smart card, this attack can target an FPGA as well. To the authors' knowledge, the first successful DPA on an FPGA was performed on a stand-alone DES [30] block cipher, programmed in a Xilinx Virtex XCV800 by Standaert *et al.* [42].

We have been able to reproduce this attack independently on a DES co-processor embedded in a SoC (10,157 logic elements and 286,720 memory bits), programmed in an Altera Stratix EP1S25. As illustrated in Fig. 4.1, we found that less than 2 000 traces were enough to retrieve the full key. The rapidity of the attack demands that adequate and sound counter-measures be researched.

4.2 Cornell Asynchronous FPGA

Various asynchronous FPGA architectures proposed in literature often use the properties of asynchronous logic for high performance (high speed, low power, robustness): (fine grain [114, 115, 113] , coarse grain [102, 110, 97], GALS [94]). Indeed, with asynchronous logic glitch-free operation and absence of clock network can substantially reduce power



(a) Logic Block (b) Bundled Data Rails (c) Pipelined Interconnect

Figure 4.2: Cornell Asynchronous FPGA

consumption, and slack elasticity [114] of asynchronous can augment the throughput.The architecture presented in this manuscript has it's focus on resistance against physical crypt-analysis, while enjoying the above-mentioned benefits of being asynchronous.

The most recently proposed asynchronous FPGA architecture [114, 115, 113] is depicted in figure 4.2. The logic block is designed assuming 1-out-2 4-phase protocol, as described in figure 4.2(a). The asynchronous LUT is designed as PCHB(see fig. 3.5.2) function implementation. The logic block also contains *carry propagation logic* and a conditional unit for *SPLIT* and *MERGE* implementation. This FPGA is particularly tuned for high speed and pipelines are used in the routing switches to increase throughput. As depicted in figure 4.2(b) routing segments are a 3-wire bundle(DATA0,DATA1,ACK). Figure.4.2(c) depicts a single switchpoint for the switchbox with asynchronous pipelines. A single *WCHB* pipeline stage is shared between tracks from four directions. The DATA wires contains two rails, and each switchpoint requires 24 switches, as compared to 6 switches(see section 2.2) for a simple subset switchbox. Although in this architecture, the interconnect balance between the rails of a dual-rail is guaranteed, since they are routed together physically, no special care has been taken to balance the logic circuits.

Our logic block architecture is much more fine grain to accommodate a plethora of encoding schemes and styles and routing architecture is single wires, on which dual rails are routed together. This is done, keeping in mind the prototyping role of an FPGA, and flexibility required for dynamic countermeasures to operate. Since we don't know of any future-proof solution to resist physical cryptanalysis, this architecture will provide the designer a soft fine grain fabric on which he/she can implement a mix of dynamic and static countermeasures pertinent to the application. In section 7.8we will discuss the additional cost to be paid for this added flexibility.

4.3 Phased Logic Asynchronous FPGA

Table 4.3 presents the statistics of phased logic implementation(see section 3.6.1). We used the basic cell(figure. 3.9) and Linder's algortihm to conduct these experiments. The table shows the no. of signals and C-Elements added by this process. The main problem of this approach is implementing asynchronous MERGE and SPLIT implementation, which calls for further modifications to the CLB architecture, and it doesn't support multi-style

Table 4.3: Phased Logic FPGA Results

Benchmark	Synchronous					Asynchronous					
	Comb	FF	nets	Tracks	Statistics	Comb	B-GATE	C-Elem	nets	Tracks	Statistics
GCD4	39	8	61	5	16 unconnected pins after synthesis, and 16 duplicate nets	39	8	4	167	12	4 gates and 25 signals added by Linder
GCD7	92	14	126	5	16 unconnected pins after synthesis, and 16 duplicate nets	92	14	5	357	16	7 gates and 46 signals added by Linder
DES Datapath	1916	184	2123	9	246 unconnected pins after synthesis, and 246 duplicate nets	1916	184	178	6350	25	31 gates and 826 signals added by Linder

asynchronous logic.

4.4 Suitability of Asynchronous FPGA for Physical Cryptanalysis Resistance

In this section we point out to the reader the motivations behind the architecture we are going to discuss. The suitability of asynchronous circuits for cryptanalysis resistance has already been investigated by [107].

- **Resistance to Fault Attacks** Random introduction of faults stalls the asynchronous circuit [106]. So the cryptanalyst doesn't receive the encrypted messages with fault syndromes. To do so, faults have to injected very carefully, at a precise time and location, which makes the attack considerably difficult.

- **Absence of a Time Reference** The absence of a reference signal (i.e CLK) in an asynchronous circuit, prevents the attacker to assume a precise model for transitions he is trying to predict, whereas in a synchronous circuit the targeted transitions must occur within the clock cycle. Moreover, the power consumption of the clock signal is clearly visible in the power trace, and provides an overall idea of the circuit operation.

- **Power constant signalling** As shown in figure 3.1(a) the supply current spikes, clearly denote the change of state of the signal, or the absence of a peak denotes that no changes occurred in the signal value. On the contrary, for asynchronous 1-out-of-2 signaling(see fig. 3.1(c)) each valid signal value is accompanied by one spike in the supply current. Note that both signals are precharged to a neutral value("00") in between the valid data. This power constant signaling falls well into the category of static countermeasures previously discussed in section 1.3.3.2.

- **Absence of Glitches** In synchronous implementations, glitches can occur without disturbing the functionality. Glitches magnify the current spikes shown in figure 3.1(a). Reference [10] discusses the effect of glitches on Side-Channel Attacks. As discussed in section 3 delay insensitive asynchronous circuits can't work in presence of glitches and consequently less vulnerable [25].

- **Reconfigurability** The motivation to opt for a reconfigurable architecture, rather than a hardwired circuit is firstly to achieve a mix between the dynamic and static

countermeasures, (see section 1.3.3) depending on the application. Secondly, as a prototyping platform for evaluating various asynchronous styles and/or masking techniques.

Part II

Design & Experiments

Summary

This part of the manuscript describes the design method, and experimental setups used for the circuits fabricated for the purpose of this PhD. Thesis as well as experimental results. Chapter 5 describes the automatic design flow for FPGA(both CMOS and reconfigurable IPs) which have been used to design the two prototypes FPGAs mentioned in this thesis manuscript. Chapter 6 discusses in detail the architecture, methods, and experiments for a fully functional Run-time Reconfigurable FPGA embedded in a SoC which is targeted for implementing counter-measures for both fault-attacks and side-channel attacks. Chapter 7 presents the architecture of a multi-style asynchronous FPGA. The routing architecture of this FPGA has been specifically designed for balanced routing and it also includes a very fast asynchronous configuration chain. This FPGA have been tested to be partially functional, and further tests are going on at TIMA, Grenoble. We propose these two FPGA architectures for protection against Side-Channel Attacks by dynamic reconfiguration(FASE 6) and protection by inherent properties of asynchronous logic(SAFE 7).

Chapter 5

Design Methodology

5.1 Introduction

Rapid evolution in technology permits us to use programmable parallel devices along with microprocessors in a SOC (System-on-Chip). While eFPGAs are flexible, and reasonably fast compared to serially executed microprocessor code, it also presents a huge cost in area compared to application specific hard IPs. Due to these factors, it is vital to have an efficient floorplan and timing budget for these programmable fabrics, which should be optimum in area and delay.

Several companies have already come up with programmable logic fabric [117, 122], but in most cases users can not customize the fabric according to their own needs. The problems associated with customized programmable logic cores are well summarized by Wilton *et al.* [91]. While [91] proposes a soft PLC (Programmable Logic Core) approach, that is the fabric synthesized as standard cell design, and can be customized by the user [51], discusses the problems of synthesizing an FPGA due to a huge number of potential timing paths. Hauck [85, 59] proposes an architecture called "Rapid" and an automatic generator of layout called "Totem". An automatic layout generator is also proposed by Kuon et. al [73].

Our CAD flow consists in integrating the bitstream and the structural VHDL generator into the place and route tool VPR [53], instead of using a separate tool. Our eFPGA architecture differs from the above mentioned architectures in these respects:

- While most other architectures propose a coarse grain reconfigurable fabric, somewhere in between a standard FPGA [85] and an ASIC, to compensate for the area and delay cost, we plan to implement a standard FPGA embedded in a SOC aimed at random logic, and modeled with VPR [53], (an open source FPGA modeling and place/route tool developed at university of Toronto).

- Our architecture is oriented towards run-time reconfigurability (RTR) and a multitasking FPGA.

These basic differences are reflected in the design flow that we are going to present.

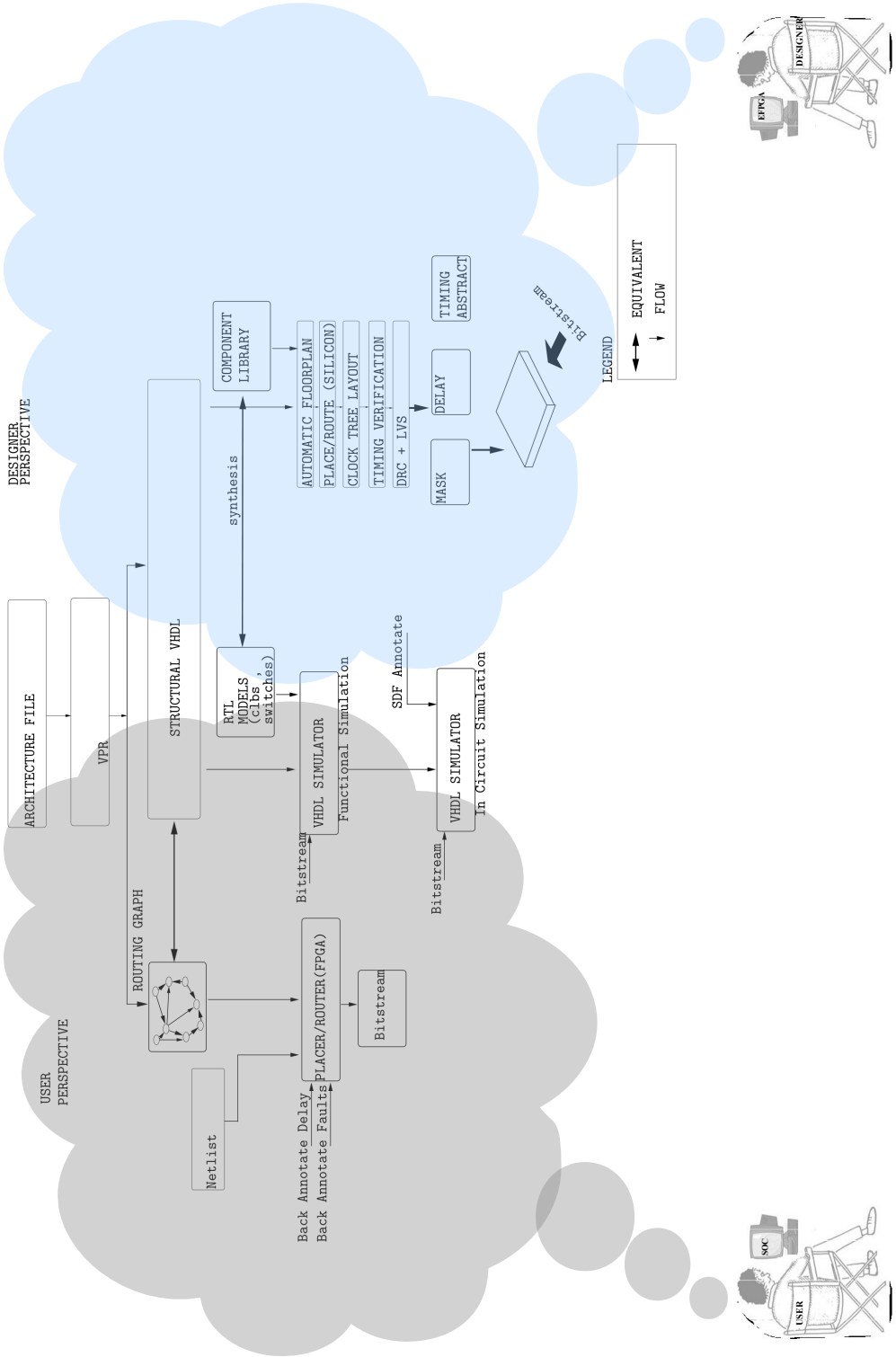


Figure 5.1: Proposed domain specific eFPGA fabric design flow.

5.2 Overview

5.2.1 Design Flow

As illustrated in figure 5.1, our FPGA design flow has a single entry point, namely the architecture file (format VPR). To fully exploit the capabilities of VPR as a FPGA modeling tool, and not as a mere Placer/Router, VPR has been modified to generate a VHDL structural model as a one to one mapping from its internal routing graph. The advantage of this flow is that, since the place/route and modeling use the same structure, the bitstream and physical layout is always coherent, and complicated modeling issues such as staggered segments, different types of switch-boxes are already taken care of by VPR.

The design flow can be viewed from two perspectives as explained in figure 5.1: the user and the designer. They share the same modeling software (VPR) and the same architecture file. A domain-specific user proceeds to design his SOC with the help of VPR, which allows him to explore the design-space, and RTL models, enabling him to rapidly simulate his dynamic reconfiguration strategies. Once a suitable architecture is determined depending on the area/time constraints in his SOC, he passes on the architecture file and the timing constraints to the FPGA fabric designer. The FPGA designer can rapidly generate a structural VHDL and an automatic floorplan from this architecture file, and provides the mask, the layout and timing abstracts to the domain specific user, which are essential for integration of the eFPGA into the SOC. Depending on the design complexity there might be a few iterations of the above transactions.

The input to the automatic layout generator is the structural VHDL generated by VPR, so it can handle FPGAs that can be modeled with VPR.

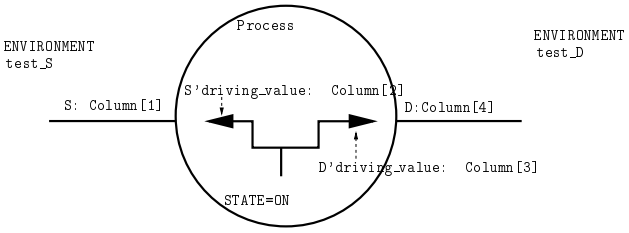
5.2.2 VHDL Model

The structural VHDL model describes the connectivity to the layout tool, and thus describes the bitstream ordering. The automatic layout generator uses this VHDL structure and generic parameters from the architecture file to repeat the same motif (i.e. the configurable tile CT which consists of the CLB, the connection boxes, and the switchbox) over space. The switchbox and connection box interconnections are described in VHDL. This model can also be laid out as a flat standard cell design, however in that case we will not be respecting the structure assumed by VPR timing analyzer.

Although this structural description does not need any synthesis, a synthesis tool is important for detecting timing violations that may occur for configuration signals w.r.t. configuration clock. To have a correct timing analysis, we disable all functional paths (i.e. paths corresponding to interconnect resources are set as false paths) and the synthesizer analyzes only the paths that are relevant for correct configuration. The functional timing analysis inside FPGA is done post-fabrication with the FPGA place/route tool (i.e. VPR). This issue is discussed in detail in section 5.4.

5.3 Simulation and Modeling of Basic Components

The basic components of our FPGA are CLB configurable logic block (4-LUT+FF), the switches (tri-state buffers, transmission gates, muxes) and the configuration points, which in our case are Flip-Flops connected in a shift register chain. Apart from the transmission gates, all other components are synthesized as a standard-cell design, thus we used the



test_S	test_D	ns	δ	1	2	3	4
0	Z	0	+0	U	Z	Z	U
0	Z	0	+1	0	Z	0	Z
0	Z	0	+2	0	Z	0	0
1	Z	10	+0	1	Z	1	0
1	Z	10	+1	1	Z	1	1
Z	1	20	+0	Z	1	Z	1
Z	1	20	+1	1	1	Z	1
1	Z	30	+0	1	Z	1	Z
1	Z	30	+1	1	Z	1	1
0	1	40	+0	0	Z	0	1
0	1	40	+1	0	X	0	X
0	1	40	+2	X	X	Z	X
0	1	40	+3	X	1	Z	1
1	0	50	+0	1	0	Z	0
1	0	50	+1	X	0	X	0
1	0	50	+2	X	Z	X	X
1	0	50	+3	1	Z	1	X
Z	H	60	+0	Z	Z	Z	X
Z	H	60	+1	1	H	Z	H
Z	H	60	+2	H	H	Z	H
Z	L	70	+0	H	L	Z	L
Z	L	70	+1	L	L	Z	L

Figure 5.2: Pass transistor model: simulated values

vendor provided models to simulate them. We are going to discuss in detail the simulation and modeling of transmission gates.

5.3.1 Pass Transistor and Transmission Gate

Pass transistors are frequently used in modern day FPGAs and reconfigurable architectures. Aside from the fact that they are more compact compared to tri-state buffer switches, they are also bi-directional. Considerable Research has been done on pass-transistor dimensions notably by Betz et al [52, 54]. However few references are available on the topic of pass-transistor modeling. Such a model seems necessary for RTR FPGAs to avoid potential short-circuits, and for validation at the early steps of the design flow, (i.e. simulation with behavioural models of other co-systems) without going into spice simulations which are too slow.

The major disadvantage with pass-transistor gates is that its output voltage can only rise to $V_{dd} - V_t$ [52]. The transmission gate circumvents this problem at the cost of one more transistor of opposite polarity.

5.3.2 Transmission Gate Model

In this section we present a transmission gate model which allows rapid simulation and functional verification of the FPGA run-time reconfiguration strategies. In figure 5.2 we depict the behaviour of this model. Either of the pins D or S can be source or sink over time. We always write a weak value (`'Z'`) to the source and the resolved value of the source to the sink. Each time the process is called (i.e. for each delta D or S changes value). Delta refers to the symbolic time for process execution. We test if the driving value is different from resolved value, to identify the source and the sink, and act accordingly. If the driving

value is different from the resolved value, then it is source, otherwise it is sink, since sink does not drive any value into the port. Pseudo code of the process is given below:

```
process(d,s)

begin
  if((d=d'driving_value and s=s'driving_value) or
  (d/=d'driving_value and s/=s'driving_value)) then
        d<= s;
        s<= d;
  else
       if(d'event and d/=d'driving_value) then
           s<=d;                                                    10
       elsif(d'event and d=d'driving_value) then
           s<='Z';
       end if;

       if(s'event and s/=s'driving_value) then
           d<=s;
       elsif(s'event and s=s'driving_value) then
           d<='Z';
       end if;
   end if;                                                          20

end process;
```

5.3.3 Automatic Floorplan

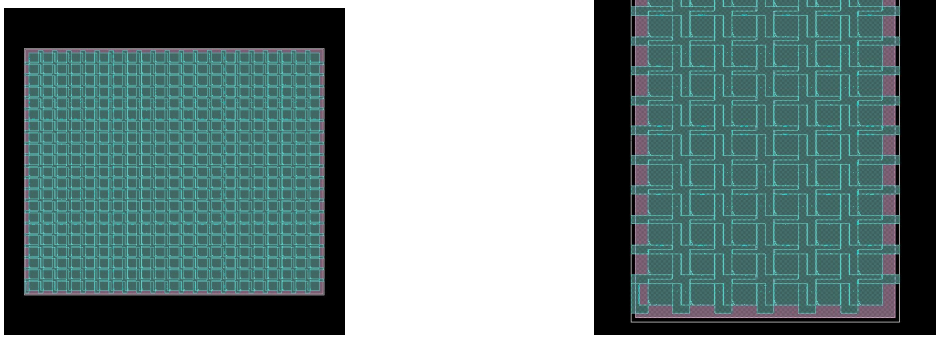
Our layout scheme is divided in two parts. Firstly the routing matrix which consists of square switch-boxes and routing channels repeated over space, and secondly the rectilinear Configurable Tiles (CLB + Connection Boxes) (see fig. 6.2) superimposed on this routing matrix. These steps are automated using Tcl/Perl scripts running on CADENCE SoC-Encounter P/R tool. The automatic layout generator in figure 5.1 works in the following manner :

- First the switches inside the switchboxes are placed and routed and a switchbox library is built. Nine different switchbox types (4 side + 4 corner + centre) are placed and routed. We use the same size for all of them because the FPGA area is mainly dependent on the centre switch box size. The routing channel ports are aligned for each switch box. These switchboxes are then placed inside the FPGA.

- Next the CTs as described in figure 6.2 are placed and routed. Their rectilinear dimensions are dependent on switchbox dimensions previously calculated. There are 5 different types of CTs (4 side + centre), as shown in figure 5.3(b).

- **clock tree synthesis:** To insert the clock tree buffers we left small slices in between the configurable tiles and switchboxes. This slice size is estimated in two passes. For the first pass all tiles are placed, and clock tree is synthesized. In the next pass all tiles are replaced with appropriate gaps between them, to insert clock tree buffers. A 'H' clock tree is generated automatically using SoC-Encounter.

- The configuration signals are routed automatically by ASIC layout tool(SOC-Encounter) in the top metal layers.



(a) W=16, 20x20, SIZE 2577.6700 μm BY 2125.4400 μm

(b) W=5, 5x11, SIZE 620.7400 μm BY 885.6000 μm

Figure 5.3: Examples of automatically generated layouts in 130 nm.

A note for architectures with long lines. Long lines are used in a staggered fashion to enhance the routability. Due to this staggering, switchbox patterns repeat themselves with a period equal to the length of the long lines. In cases where different lengths are used the period is the LCM of these lengths. In our flow, since the VHDL code generated is an exact replica of the routing graph, each switchbox is described individually, and can be laid out exactly as it is seen by the modelling tool(VPR). However for bigger FPGAs this could become excessively time consuming for tools that handle this detailed VHDL code. To avoid that, once the VHDL code is generated, we only keep the useful part and repeat it with a frequency equal to the LCM of the lengths. The Logic element and connection box patterns are the same, and hence it can be repeated with a frequency of 1.

More details about simulation, rectilinear floorplan etc. can be found in [137].

5.3.4 Comparison with Previous Works

Reference [74] describes an automated FPGA layout generator(GILES) based on VPR. To compare it with our approach, we generate the layout using the same architecture described in [74]. This GILES reference architecture has been laid out with 180nm, so we scale it down to 130nm and do the comparison(see table 5.1). While comparing it should be kept in mind that the basic cells used are considerably different. While Giles used SRAM cells, we have used D Flip-Flops as our basic configuration point. We also point out the difference in the size of buffer switches used in both cases in table 5.1.

Figure 5.3 shows two example floorplans generated with our flow in 130nm process. W is the channel width used, and units are in microns.

5.4 System Integration

5.4.1 Timing Abstract Generation of eFPGA

Generating a timing and layout abstract of the eFPGA is like the data-sheet of a conventional FPGA. Automatic timing analyzers consider all possible paths in a design, to determine the critical path. Obviously in an FPGA all paths are not of interest, e.g. the

Table 5.1: Comparison with the GILES Reference Architecture.

Flow	**Tile Area**	**Tech - nology**	**Switch Area**	**Memory cell type**	**Memory cell Area**
Giles Flow	48,282 μm^2	180nm	18.30 μm^2	**4x4 SRAM**	13.07 μm^2
Giles Flow (Technology scaled)	24,141 μm^2	130nm (Technology Scaled)	9.15 μm^2	**4x4 SRAM**	6.53 μm^2
Our Flow	28,009 μm^2	130nm	14.12 μm^2	**D FF**	40.3 μm^2

paths from configuration input signals to the functional outputs. Another example is a path in the FPGA which includes all the switches, since we are not going to program the FPGA in such a manner. So the following method has been used to generate the timing abstract of the eFPGA.

- First we fix the critical path delay t_{crit} inside the FPGA. This depends on the period of the clock signal used in SOC ($t_{crit} = t_{clk\_soc}$). This must be compared beforehand with the timings of a set of benchmark circuits placed/routed in FPGA. In our case the clock period is 15 ns. We also fix the maximum input delay for all input pins, and maximum output delay for all output pins.
- While generating the timing abstract, we disable all functional paths inside the FPGA, and all paths between configuration and functional signals.
- For each functional inputs we add only one path to each of the outputs with a delay t_{crit}.
- All the configuration signals are synchronous, so they are not disabled, and we specify the transition time for each signal.
- We include the clock insertion delay in the timing abstract, which is necessary for clock tree generation at the top level.

The timing abstract permits the ASIC timing analyzer to optimize the critical paths that are partly inside the FPGA and partly in SOC. The value t_{crit} is then passed on to FPGA place/route and timing analyzer tool (VPR) which guarantees this delay inside FPGA.

5.5 Conclusion

Taken into consideration the cost in area and delay of an eFPGA, a customized eFPGA is inevitable, since any unused costly resource might not be acceptable for domain specific users. This flow proposes enhancements for frontend, backend as well as integration into SOC. The planned major enhancements to this flow is to enlarge the design space to include several other types of FPGA architectures (e.g. hierarchical FPGAs) which may be customized according to the user's needs. Integrating the automatic layout generator to standard tools will be an added functionality for SOC designers. The two prototype

FPGAs mentioned in this thesis manuscript namely (FASE and SAFE) are designed and implemented with this automated flow, with very little customization necessary for the different nature and configuration scheme for these two FPGAs.

Chapter 6

A Run-Time Reconfigurable Architecture: FASE

6.1 Introduction

The hardware reconfigurability is considered when applications are constrained by a high degree of both flexibility and performance. Many academic projects studied RTR architectures, to speed up reconfiguration time and increase the flexibility for reconfigurable computing [88, 96, 89]. Some commercial FPGAs provide dynamic reconfigurability features. XILINX offers partial reconfigurability by columns in its VIRTEXII offer [134]. ATMEL proposes the AT40K and an embedded FPGA in the FPSLIC family based on AVR microcontrollers [125]. Many academics studied optimized tools for fast reconfiguration [63, 66]. However, most innovative architectures like the ATMEL AT6000 and XILINX XC6200 did not get any commercial success and disappeared. With new emerging challenges like security, reconfigurable architectures could have a second wind. Reference [82, 83] presents and evaluates coarse grain reconfigurable architectures secure agianst DPA, as opposed to our fine grain architecture.

Reconfigurability is a good strategy to secure cryptographic accelerators, that are targets of side-channel or fault injection attacks. We suggest, for instance, that a permanent and random change of the configuration is able to conceal side-channel information from the attacker. The main drawbacks of existing reconfigurable architectures are twofold:

1. The architectures are proprietary with many unknowns in the routing structure and reconfiguration hardware.

2. The proposed methods and tools appear to be very constrained by the architecture without improvement possibility.

The XILINX architectures are the most constrained because of their coarse-grain column-wise approach. The ATMEL has a fine-grain approach with a reconfiguration at the cell level. However, their legacy Place/Route tool (FIGARO [84]) imposes a static placement of the reconfigured area.Therefore, it seems quite challenging to evaluate the robustness of FPGAs and propose tamper resistant circuits without a close collaboration with the manufacturer. In the security context, the FASE "FPGA Architecture for Secure Embedded systems" project aims at designing an FPGA architecture with RTR capability which meets these requirements:

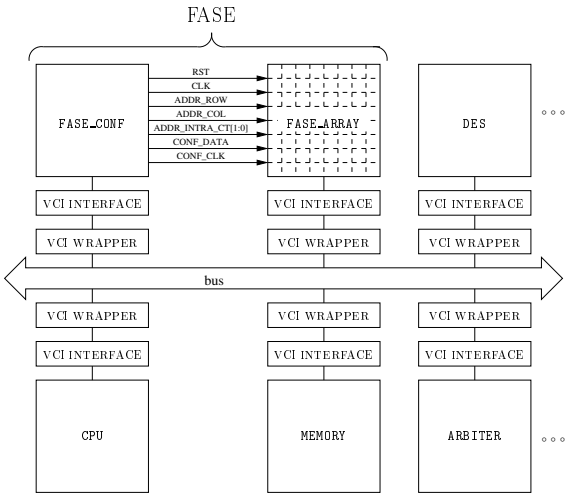


Figure 6.1: VCI interface for `FASE_ARRAY` and `FASE_CONF`.

- A fine grain architecture which allows the designer to place/route dynamically the reconfigured area anywhere and at cell boundaries.
- An open architecture with detailed and exhaustive specifications of the routing and logic resources.
- A simple configuration interface which allows the programmer to build his own reconfiguration strategy.

FASE is an offshoot of the SAFE (Secured Asynchronous FPGA for Embedded systems) project [120] which takes profit of asynchronous cells to increase the robustness against side-channel attacks. This project specifies that the partial and dynamic configuration has to accommodate several modules possibly implemented in different styles of asynchronous logic [99]. It should also be able to modify the design on-the-fly on detection of intrusion. By contrast, FASE is a synchronous FPGA, thus intrinsically unsafe against side-channel attacks. However, its dynamic reconfiguration capability also enables it to implement both preventive and resilience strategies (that are not built-in.)

The rest of the manuscript is organized as follows. Section 6.2 presents the main principles of FASE functional and configuration architecture. Section 6.3 addresses the issues involved in reconfiguration and presents a typical design flow. In section 6.4, two applications requiring a high level of security are presented. Finally, section 6.11 draws the conclusion.

6.2 FASE Architecture

6.2.1 FASE Overall Architecture and Principles

As an embedded FPGA, FASE is designed to be connectable to a system bus. It thus features a VCI [123] interface. The general architecture of FASE comprises of a functional array (`FASE_ARRAY`) and of a configuration controller (`FASE_CONF`), as shown in figure 6.1.

FASE has a generic architecture described in the sequel as per the VPR (Versatile Place-and-Route tool [53, 54]) nomenclature. However, for the sake of illustration, fixed valued

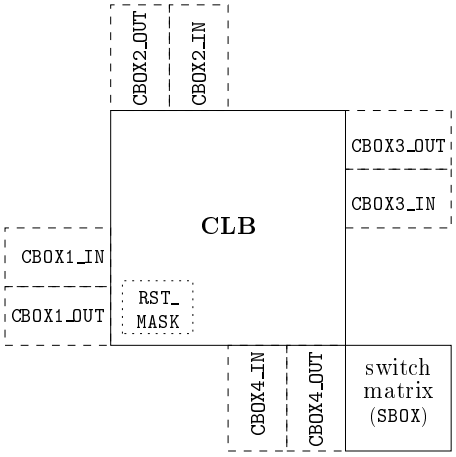


Figure 6.2: The configurable tile (CT) and its components.

are given to some structural parameters. The FASE array (FASE_ARRAY) is a reconfigurable embedded architecture based on four hierarchical objects:

1. The logic element (LE) is composed of a look-up table (LUT) and a D-flip-flop (DFF). A reset mask RST_MASK indicates whether the DFF reset line is active or not.

2. The compound logic block (CLB) is composed of several LEs (only one in the sequel.)

3. The configurable tile (CT), depicted in figure 6.2, is composed of one CLB plus, at its periphery, the following switching components:

   - The connection box CBOX_IN (resp. CBOX_OUT) permits the CLB input (resp. output) connections.
   - The switch box (SBOX) allows the routing between CLBs.

4. And finally, the array FASE_ARRAY is an $N \times N$ square set of CTs surrounded by $4 \times N$ CTs dedicated to the I/Os (IOBs.)

The configuration controller (FASE_CONF) is in charge of configuring and initializing any area of the FPGA.
The RTR in FASE is based on two configuration levels:

1. first of all, the selection of a specific set of CTs or IOBs amongst the array, and

2. secondly, the CT internal components: CLB (with the RST_MASK), CBOX_IN, CBOX_OUT and SBOX.

A configuration zone corresponds to a set of CT, not necessarily rectangular. The RTR can be applied by reconfiguring subsets of objects inside a subset of CT. For instance, only the RST_MASK of the configured area is cleared (so that it is properly initialized) while the rest of the FASE_ARRAY remains unaffected by the global reset (because the RST_MASK is set.) Detailed examples are given in Sec. 6.3.

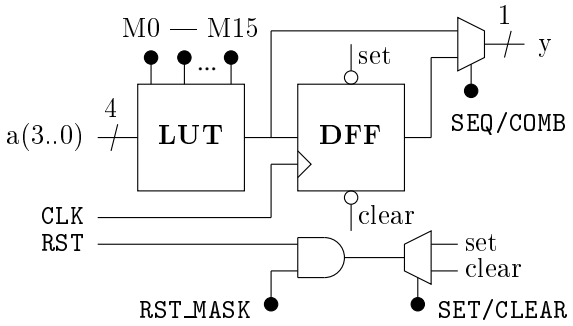


Figure 6.3: CLB comprised of (only) one $4 \rightarrow 1$ LE with maskable reset.

6.2.2 Functional Architecture

The FASE architecture presentation does not insist on any particular performance improvement. The reason is that most optimizations and trade-offs published in the literature can be transposed in a straightforward way to FASE simple and generic structure.

6.2.2.1 CLB

In addition to the LUT mask, a configuration point SET/CLEAR selects whether the flip-flop should be set or cleared when RST is active. The configuration point SEQ/COMB selects between the "LUT only" or the "LUT + DFF" functionality. The RST_MASK is added to selectively initialize CLBs in FASE. This configuration point is addressable independently of the CLB configuration chain. Figure 6.3 illustrates a CLB composed of one $4 \rightarrow 1$ LE with maskable reset. In all the figures, a solid dot (•) represents a configuration memory point.

6.2.2.2 Routing Resources

To simplify this section, all the routing tracks are segments of unity length and use a uniform channel width W. We design the input and output connection box flexibilities to be 50% (i.e. the CLB inputs and output connect to 50% of the routing tracks) and we use a Wilton switch box [91] to achieve greater routability. The IOB flexibility is unitary. The global signals RST and CLK are routed separately on dedicated tracks.

6.2.2.3 Functional Interface

FASE_ARRAY is linked to the external world via the VCI interface which contains a dual-port RAM accessible by both FASE_ARRAY and VCI. Four IOBs are dedicated to control signals:

- CMD is used by the VCI interface to start an operation.
- EOC is set by FASE_ARRAY to signal the operation end.
- RAMEN indicates that FASE_ARRAY currently accesses the RAM.
- WE indicates that FASE_ARRAY writes into the RAM.

$4 \times N - 4$ IOBs are available as address and data lines to access the dual-port RAM. For instance if $N = 8$, 12 IOBs pads could compose the address word and 16 could be the data (8 inputs and 8 outputs). The details of the functional interface is depicted in figure 6.4.

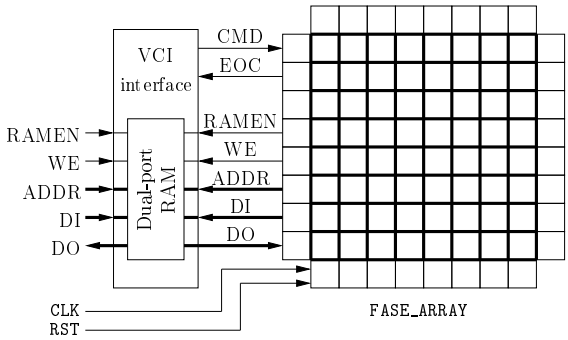


Figure 6.4: VCI interface of the FASE_ARRAY with functional IO pads.

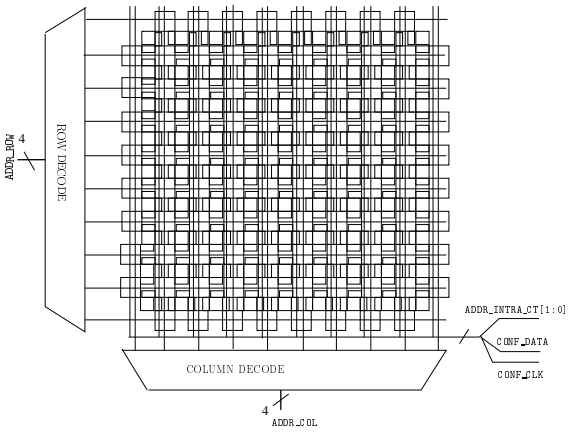


Figure 6.5: Address lines and global configuration signals.

6.2.3 Configuration

The configurable memory points are programmed via a set of shift registers inside each CT. From a configuration viewpoint, an IOB is considered as being a CT subset. This is because the IOB has no CLB and the number of CBOX and SBOX depends on the IOB location. At power up, the power on reset signal (denoted PO_RST) permits to start with all the configuration points inactive.

6.2.3.1 Configuration Architecture

In FASE, each CT and IOB is addressable by the ADDR_ROW and ADDR_COL lines, as illustrated by figure 6.5.

Inside a CT, there are four configuration chains selected by the signals ADDR_INTRA_CT[1:0]. Each chain corresponds to specific CT components:

1. CBOX_OUT,
2. CBOX_IN + CLB,
3. SBOX and
4. RST_MASK.

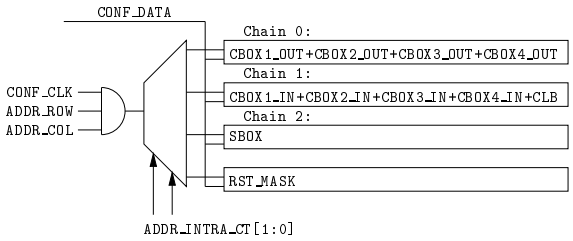


Figure 6.6: Separately addressable configuration chains of FASE.

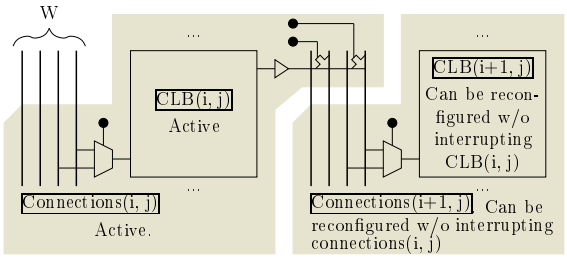


Figure 6.7: Input connection box: granularity of configuration (W=4, fc_in=fc_out=$\frac{1}{2}$).

To avoid electrical conflicts due to shifting of configuration bits along the chain, the CBOX_OUT is disabled during the CT configuration period, except if the RST_MASK is being configured. This will allow the designer to split dynamically active blocks and inactive blocks without any conflict or operation interrupt.

The chain input is CONF_DATA and the chain clock is CONF_CLK.

Figure 6.6 depicts the architecture of the four configuration chains.

The configuration points drive logic directly inside the CLB or drive pass-transistors for the connection boxes and the switch box. To save a few configuration bits, the connexion boxes use multiplexer switches. The output connection boxes use tri-state buffers rather than pass-transistors to allow high fan-out drive. Figure 6.7 shows the connection box configuration points.

If we consider one LE per CLB, the number of configuration points as in figure 6.3, the number of configuration points is as follows:

- CLB: 18 (M0 – M15, SEQ/COMB, SET/CLEAR),
- RST_MASK: 1,
- CBOX_IN: (W × Fc_in) × number of inputs,
- CBOX_OUT: (W × Fc_out) × number of outputs,
- SBOX: 6 × W,

where: W is the number of tracks per row or column, Fc_in and Fc_out are respectively the flexibilities of the CBOX_IN and CBOX_OUT.

6.2.3.2 Configuration Interface

FASE_CONF is in charge of the configuration and of the delivery of global signals that are CLK and RST. Like FASE_ARRAY, it is connected to the external world via a VCI interface. FASE_CONF generates the signals described in table 6.1:

Table 6.1: Interface between FASE_CONF and FASE_ARRAY.

Signal	Function
RST	Global reset. It can be masked for each the CT by the RST_MASK bit.
CLK	Global clock. The clock frequency can be adjusted in FASE_CONF to satisfy the timings.
ADDR_ROW	Row address. Selects a row among $N + 2$ for configuration.
ADDR_COL	Column address. Selects a column among $N + 2$ for configuration.
ADDR_INTRA_CT[1:0]	2 bits CT component address. Selects a specific component set inside a CT.
CONF_DATA	Configuration Data. One bit data to enter the configuration chains.
CONF_CLK	Configuration clock.

Table 6.2: Instruction set for FASE_CONF.

Instructions	Function
SET_ROW <ROW_ADDR>	Selects cell address.
SET_COLUMN <COL_ADDR>	Selects cell address.
CONFIG_C_OUT <CONFIG_DATA>	Configures input connections.
CONFIG_C_IN <CONFIG_DATA>	Configures output connections.
CONFIG_SBOX <CONFIG_DATA>	Configures switch-box connections.
ENABLE_RESET	Enables/Masks the RST_MASK register.
DESELECT	Deselects everything.
START	Generates a reset.

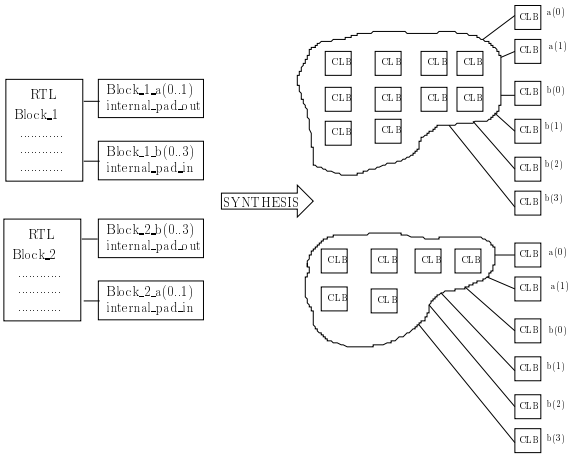


Figure 6.8: Design flow outline.

All the signals are global, except `ADDR_ROW` and `ADDR_COL` that are decoded and associated with respectively a specific row and column as shown in figure 6.5.

The `FASE_CONF` reads the instructions and configuration data from the RAM and sends a serial bitstream to the proper address in `FASE_ARRAY`. The basic instruction set is given in table 6.2.

6.3 Run-Time Reconfiguration

RTR soft IPs or DHPs [66] can be called any time into FASE at the presence of already active blocks. This can be achieved by meeting specific RTR rules and methods at 3 hierarchical levels: application, circuit and block.

6.3.1 RTR at Application Level

Dynamic resource management [90] is necessary to efficiently use RTR. RTR modules should be "allocated" before their configuration and "freed" once they are no more in use. This alloc/free information is passed on to the FPGA compilation tool during run-time. This step is necessary as new incoming block depends on the present occupation status of the FPGA. Incidentally the synthesis/place/route tool is analogous to the compiler/linker for microprocessor based systems.

6.3.2 RTR at Circuit Level

The circuit design flow needs to integrate specific interfaces between blocks in order to allow flexibility for the RTR. For this purpose, dedicated `CT` have to be used as internal IO pads. At the RTL level, they correspond to the entity inputs and outputs as shown in figure 6.8.

This netlist is then placed/routed with VPR. The simulated annealing algorithm in VPR [79] may generate an arbitrary shaped placement in order to minimize the routing resources. However, we constrain the placement such that the `CLB`s configured as internal pads are always placed at the frontier and the placement avoids the cells already in use as depicted in figure 6.9.

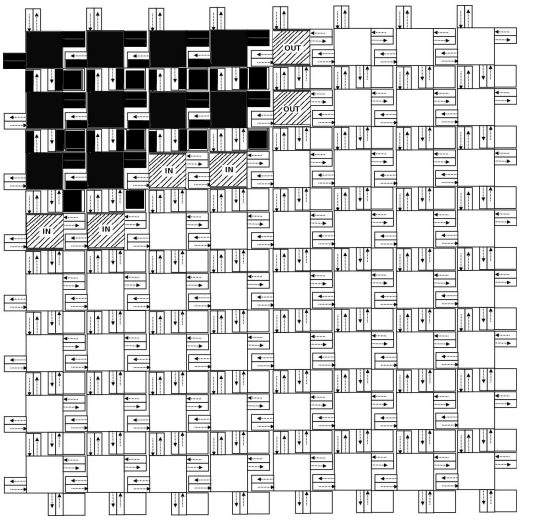


Figure 6.9: Block_1 (black) active, CLBs configured as internal pads are placed at the boundary.

Synchronization issues at initialization between active soft IPs and a newly loaded module is left to the *top_level* RTL designers, for the sake of flexibility. Whether the new incoming block is "ready" or "not ready" can be communicated to the active blocks by many protocols. For example, it can consist in polling the status of the new IP block written into RAM, or one of the active IPs may initiate the configuration of this new block.

An overview of the RTR design flow is described in the following sequence:

1. RTL code is synthesized to generate soft IP cores connected by CTs configured as internal pads.

2. Block_1 is configured into FASE and initialized, all the internal pads are constrained to be placed at the boundary of Block_1.

3. The placer/router loads the current occupation status of the FPGA and then Block_2 is placed and routed. It is constrained to avoid the already occupied CTs. This can be done in many ways in VPR. One simple method is to set the cost functions of already occupied CLBs equal to infinity. During the placement/routing of Block_2 the router takes into account the positions of the internal pads.

4. Block_2 is configured into FASE and the internal pads are configured to connect it to Block_1.

5. Block_2 is selectively initialized.

Figure 6.10 shows the floorplan result.

6.3.3 RTR Rules at Block Level

6.3.3.1 Timing closure

For timing closure placement/routing, a simple strategy consists in considering a safety margin greater than the worst case clock period for the entire system. This basic approach

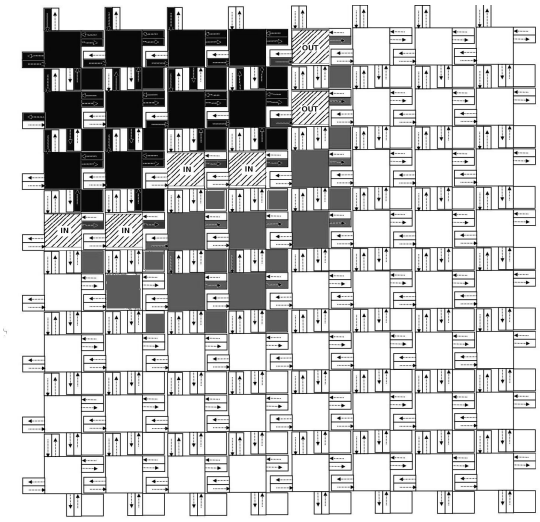


Figure 6.10: Block_2 (gray) is configured, internal pads are configured to connect to Block_1.

will no more be necessary for future FASE release which will use asynchronous CLBs. The self-timed property of asynchronous calculation will remove this constraint intrinsic to synchronous circuits.

6.3.3.2 Initialization

Soft IPs have be selectively initialized in FASE without interrupting others. The way to send the RST only to the block which has to be initialized is to use the RST_MASK configuration point. While configuring new RTR blocks into FASE, the RST_MASK of all active CLBs is masked so that only new configured blocks are initialized.

6.3.3.3 RTR Sequence

To avoid conflicts during the shifting of the bitstream, each time one of the four chains is selected for configuration, the outputs of the configured CT are disabled. Only the RST_MASK configuration point can be programmed without disabling the outputs. Let us assume that Block_1 is active and Block_2 is being programmed.

The sequence of reconfiguration of Block_2 is illustrated in figure 6.11.

Block_2 could have a greater or smaller size or even be at a new location. In this case the CBOX_OUT of the old location have to be disabled before configuring the new block.

Each time a RTR soft IP is freed from the system, all configuration bits in the CT belonging to the freed block are set to 0.

6.4 FASE Architecture Suitability for Security Applications

The key feature of FASE is that it enables the design of robust implementations, which indeed demand an unrestricted access to configuration resources.

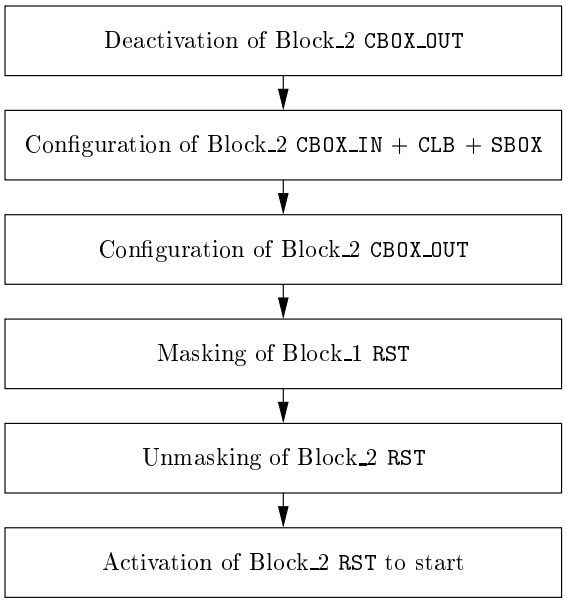


Figure 6.11: RTR sequence.

6.4.1 Security Requirements on Hardware

The security applications are encountering the problem of the proliferation of cryptographic algorithms. Until the years 2000, the smart cards could perform all the security functions (authentication and encryption) thanks to DES and RSA. The security of those two algorithms is currently questioned, and many alternatives are put forward. Since 2001, DES is officially superseded by the AES, but other candidates are promoted (for instance Khazad in Europe.) The same applies to asymmetric encryption primitives: the regional variants of the digital signature algorithms are numerous.

The smartcard industry is thus facing a dilemma: the devices are either cost-efficient or interoperable. The most viable solution thus consists in enabling an applicative agility within the smartcard, using an *ad hoc* e-FPGA. This way, virtually any algorithm can be implemented with hardware acceleration support: only the credentials (key, seed, *etc.* material) need to be resident into the smart card, the algorithms being either downloaded or programmed on-the-fly from an internal ROM. The FASE architecture enables pervasive reconfigurability by the use of hardware accelerated mobile code.

In addition, security hardware must also protect itself from implementation-level malicious attacks. The security environment is indeed harsh for embedded systems. Embedded system cannot afford tamper protection used otherwise for critical equipments. It must thus be assumed that invasive attacks [3] are likely to be used against those systems. The security requirements are becoming very stringent: the two examples of sub-sections 6.4.2 and 6.4.3 illustrate how RTR can provide an elegant solution to the otherwise difficult problem of mixed passive/active threats [118].

6.4.2 FASE Usage for DPA-proof Hardware Accelerators

The side-channel attacks [22] consist in monitoring the instant power or electromagnetic emissions of a device in order to extract information from the computation internals. The typical protections against those attacks basically boil down to doubling either the execu-

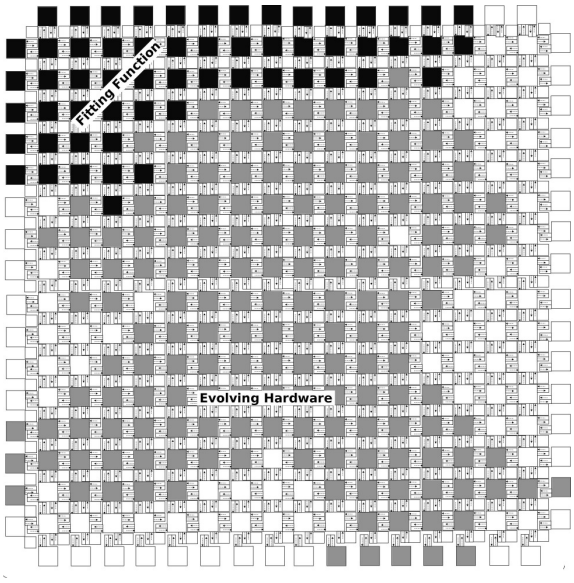


Figure 6.12: Example of the use of RTR to achieve a DPA-proof logic block.

tion time (in the case of software) or the implementation area (in the case of hardware) [12]. We propose to use FASE random-access RTR capability to modify the algorithm implementation at every invocation. This strategy makes it impossible for an attacker to collect consistent traces, thus discarding the DPA fundamental hypothesis that execution symptoms can be accumulated for statistical treatments. The implementation mutations can be, in this example of DPA protection, controlled by a random number generator (RNG.) A snapshots of an example DPA protected applications is depicted in figure 6.12: the fitting function (RNG) is constant and always active and the evolving portion (DPA-proof cryproprocessor) is reconfigured continuously.

6.4.3 FASE Usage for FA-proof Hardware Accelerators

Fault attacks (FA) [6] intentionally disturb a cryptographic algorithm so as to extract information from the faulty executions. Usual counter-measures against this class of attack consist in adding redundant hardware to detect and possibly correct the faults. However, the main drawback of this approach is that the additional hardware is useless if no fault or light attack occurs. We propose to take advantage of FASE RTR to implement a graded and adaptive faults detection capability. This strategy allows for a cost-effective survivability strategy: as the environment becomes more aggressive, the algorithm implemented in FASE improves its detection codes. In figure 6.13, a memory management unit (MMU) interfaces a FA-resistant core with the rest of the SoC. The MMU is always active whereas RTR soft IP cores (*e.g.* DES, RSA using a Montgomery Modular Multiplier) can be loaded into the system on demand.

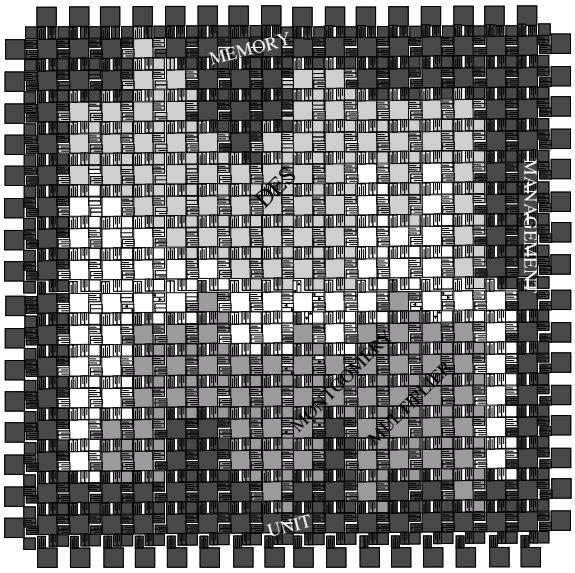


Figure 6.13: Example of the use of RTR to achieve a FA-proof logic block.

6.5 Implementation Details

6.6 Interface

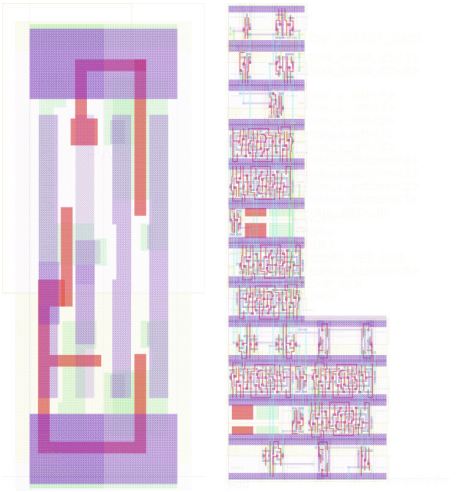
During boot process the eFPGA is unprogrammed. Unknown values for critical output signals connected to the interfacing circuitry, might cause a failure during boot process (e.g. an interrupt line is asserted). To avoid that, at Power-on-Reset every output signal is disconnected from the internal cells of eFPGA, and pulled up/down to a known value. During normal operation these outputs must be reprogrammed to their original functionality.

6.6.1 Architecture and Layout Summary

Our SOC architecture is based upon an open source microprocessor (FREE6502), and several cryptographic IPs resistant against side channel attacks. It also includes a dynamically reconfigurable FPGA code named FASE. The system architecture (figure 6.1) is discussed in detail in 6.2. The SOC architecture is based on industry standard VCI interface. The eFPGA FASE consists of `FASE_ARRAY` which is the island style gate array structure and `FASE_CONF`, the configuration controller as described in figure 6.1. `FASE_CONF` is a simple state machine, with a small set of instructions to describe the reconfiguration sequence. Apart from the VCI interface, there is a dedicated interface between `FASE_ARRAY` and `FASE_CONF` which consists of configuration signals and the configuration clock.

Figure 6.15(b) shows FPGA automatic floorplan, with all routing channels aligned. The relative position of CTs and swithboxes are highlighted. It occupies an area of 0.6 mm^2. Figure 6.14(f) shows the placement and routing inside a rectilinear CT. Our SOC SECMAT has been laid out in ST-Microelectronics 0.13 μm 6-layer process. The SOC micrograph (4.1 mm^2) is depicted in figure 6.16 with the 8×8 embedded FPGA. The configuration controller `FASE_CONF` is dissolved in glue logic.

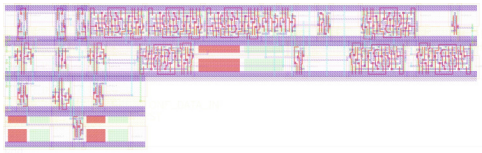
Figure 6.16 shows the micrograph of the SoC SECMAT. All the components on the



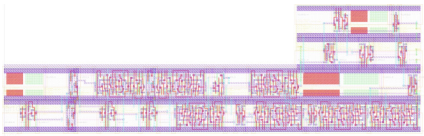
(a) Transmission Gate Standard Cell

(b) Tile:South

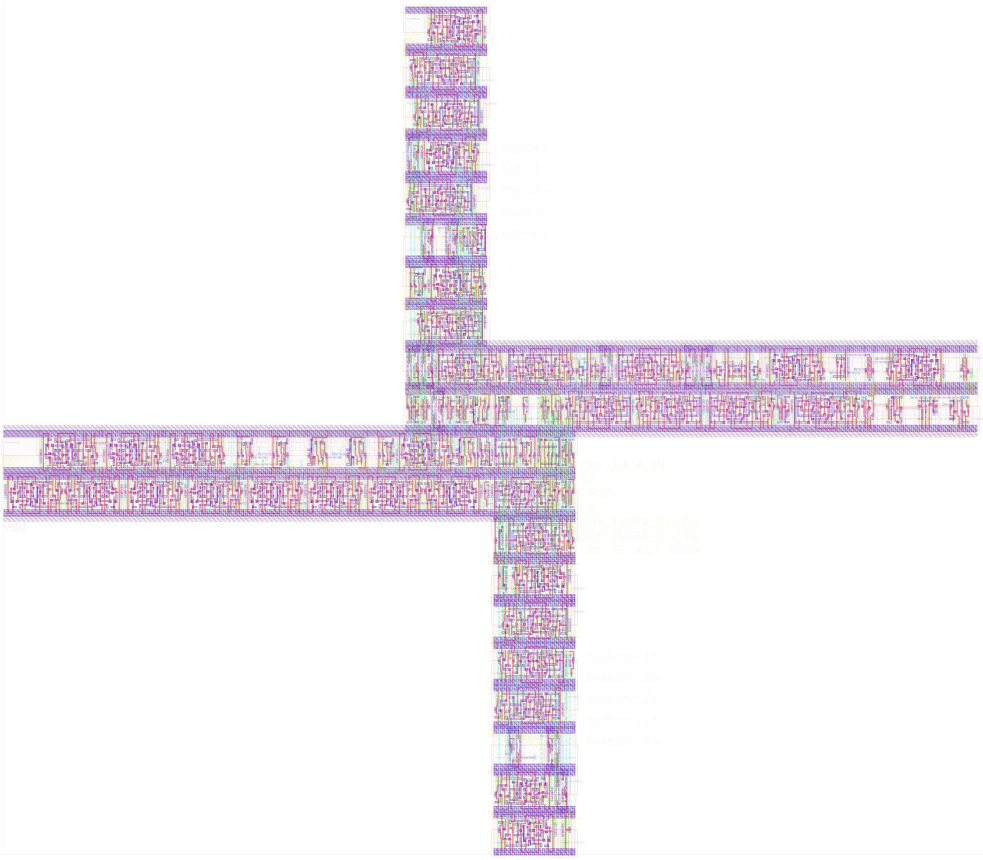
(c) Tile:North



(d) Tile:East

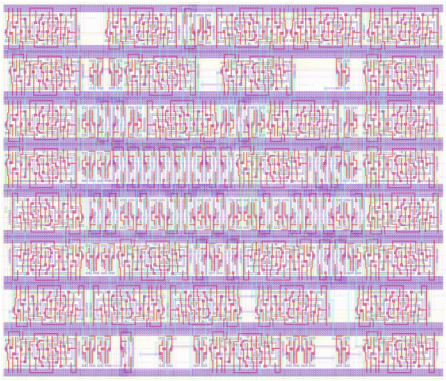


(e) Tile:West

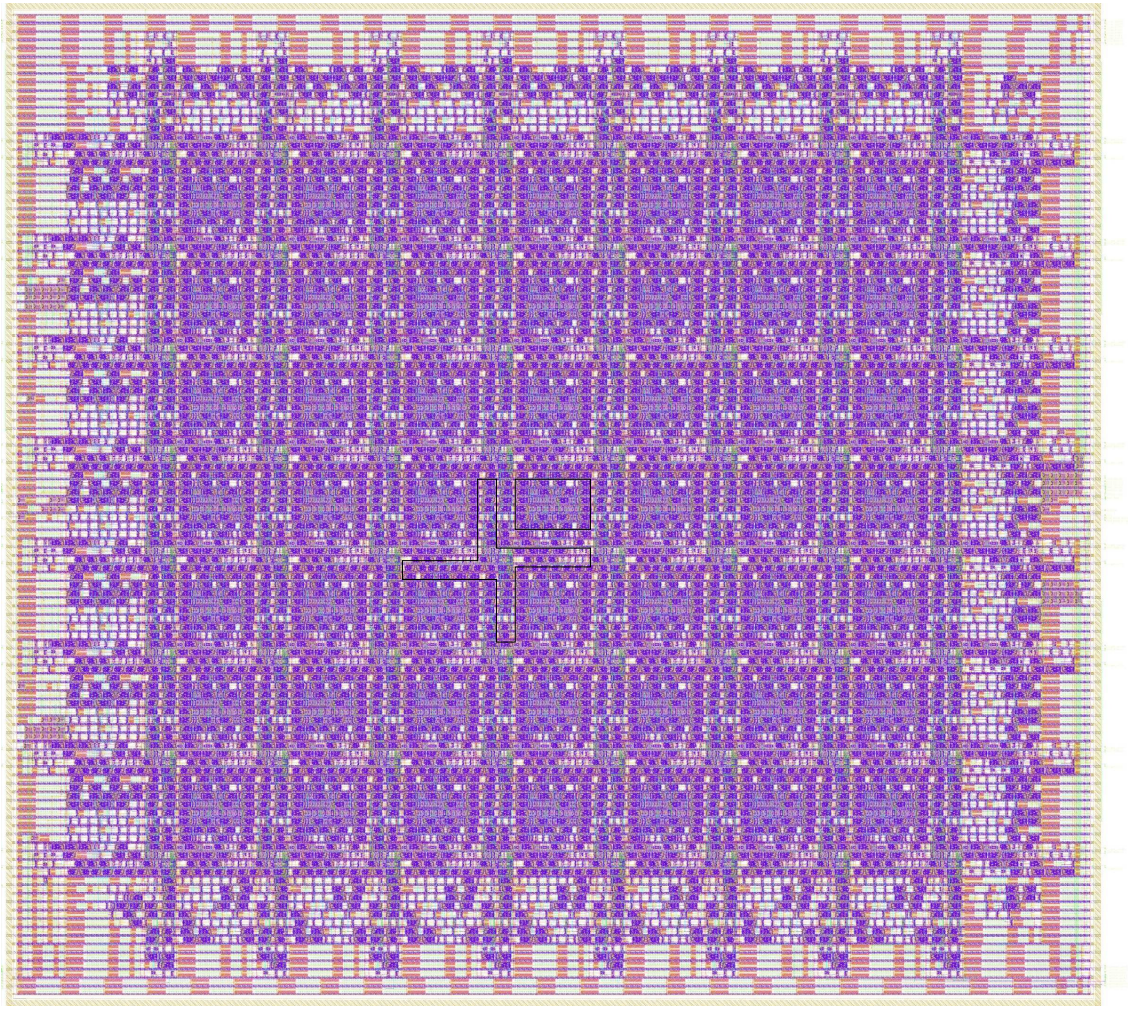


(f) Tile:Centre

Figure 6.14: FASE Layout details I



(a) SwitchBox



(b) Fase Array

Figure 6.15: FASE Layout details II

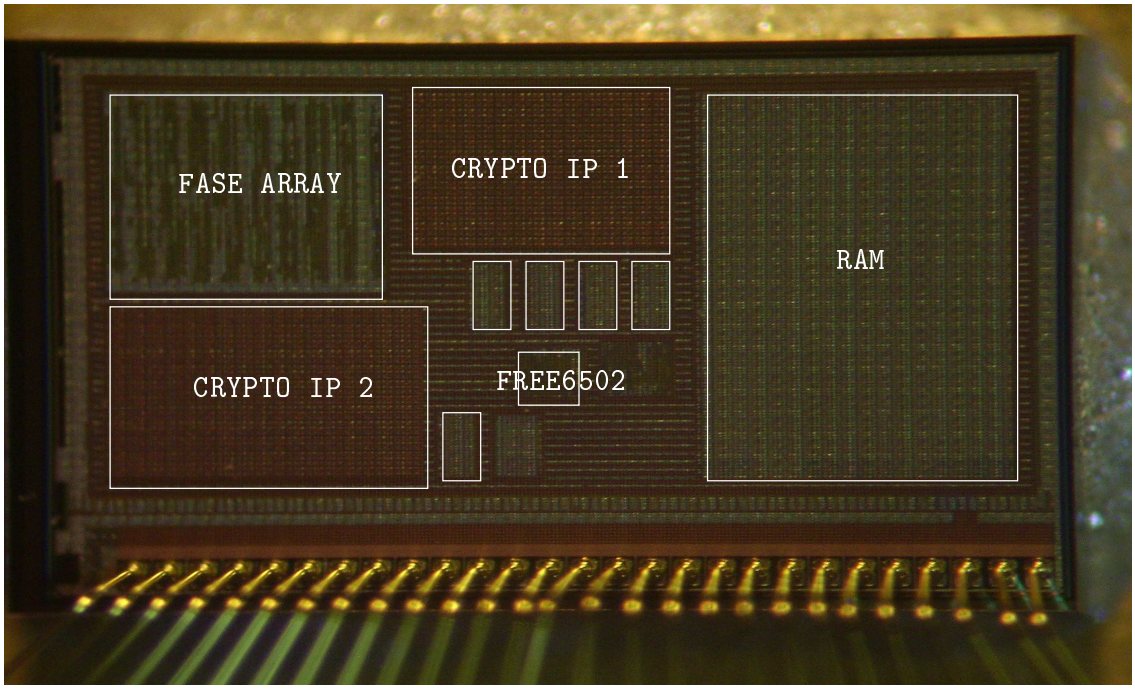


Figure 6.16: SOC micrograph, eFPGA in the upper left corner.

Table 6.3: Physical Details.

Die Size(SoC)	$4.1mm^2$
Area(FASE ARRAY)	$0.6mm^2$
Transistor Count	200,000 (Approx.)
V_{dd}	1.2V
V_{dd} IO	1.2V
F_{max}	32MHz

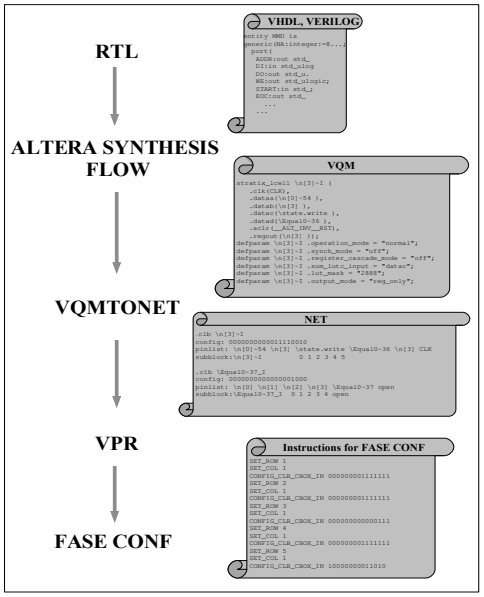


Figure 6.17: IP design flow.

VCI bus are highlighted. The configuration controller is in the glue logic around FPGA. The small highlighted block are the RAMs for each VCI module, VCI interface is in the glue logic. Our circuit have been fabricated with ST Microelectronics 130nm 6-layer process(CMP run SI2C7_1). Various physical details about the FPGA are listed in table 6.3. Although the configuration circuitry in the FPGA has been designed to operate at 66MHz, we have tested the correct functionality upto 32MHz.

Figure 8.4 in appendix shows the test setup for our SoC SECMAT. All the input/outputs of SECMAT are controlled by an interfacing card with on board Linux [126]. Programs are downloaded into SECMAT via this card which is connected to Ethernet. Each VCI module in SECMAT has its own power supply pins for accurate power measurements.

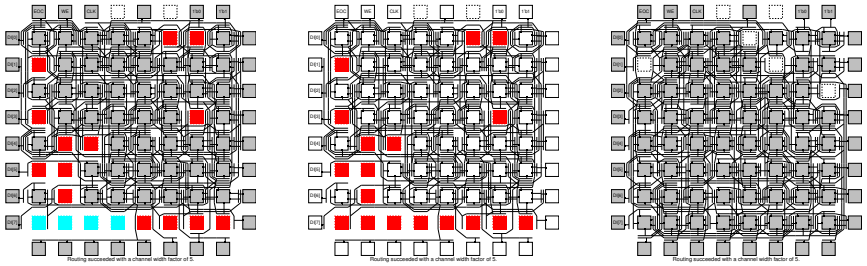
6.7 IP Design Flow

The design flow to develop user IPs are outlined in figure 6.17. Our flow is based on industry standard Altera [67] synthesis tools. The VQM netlists generated by Quartus synthesis are transformed to the VPR net format with `vqmtonet` which is a lex/yacc based parser. We added one more field to the VPR netlist format which contains the bitstream for each CLB. After place/route the modified VPR generates the bitstreams as a set of instructions for `FASE_CONF`.

6.8 Experiment

The dynamic IPs concerned are

- IP1: Does a cyclic Vernam cipher (Xoring the message with a pre-defined key) with the 256 bytes in the RAM. The key is stored in the IP itself.
- IP2: A 24 bit counter, which stores the count value in the RAM.



(a) IP2 in execution and IP1 CLBs & switchboxes are configured (red ■ & cyan ■).

(b) IP2 erased.

(c) IP1 functional.

Figure 6.18: Run-Time Reconfiguration of IP2 while IP1 is in execution.

Table 6.4: Demonstration of RTR on 8x8 FPGA.

Hardware Blocks	**No of CLBs**	**Execution Time**	**Reconfig Time**	**Erase Time**	**Reconfig Ratio**
	LUT4+FF	(Computation Cycles)	(Computation Cycles)	(Computation Cycles)	$\frac{\text{Column III}}{\text{Column IV}}$
IP1	58	1024	3712	174	0.276
IP2	44	67108864	2816	132	23831.27

The goal of this experiment is to validate the correct functionality in presence of run-time reconfiguration and to find out the bottlenecks of such systems.

The experiment itself consists in

- First configuring IP2 into the FPGA.
- During the execution of IP2, partially configuring IP1 into the CTs that are not used by IP2.
- Once IP2 finishes, erase IP2, full configuration and execution of IP1 .
- Verify that the results are correct.

Table 6.4 provides the various details of this experiment. Figure 6.8 illustrates the various steps of the experiment. In Fig 6.18(a) the places marked with cyan means both the CLB, inputs and the switchbox are being configured, red means only the CLB and inputs are being configured.

The difference in `reconfiguration time` and `Erase Time` is due to the fact that, while erasing only the output connection of CLBs are erased to avoid any short circuit during reconfiguration. After that, the configuration for input connection boxes, CLBs and Switch Boxes are overwritten with the new configuration.

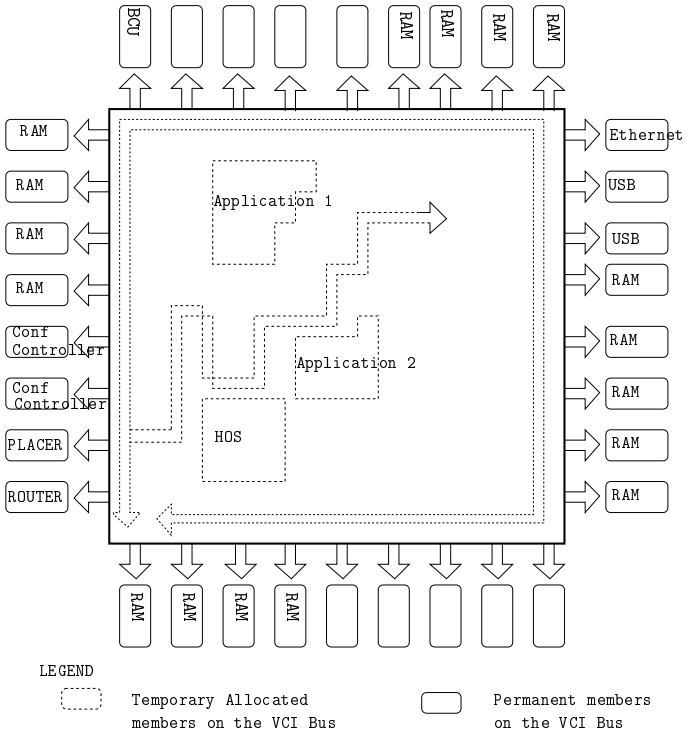


Figure 6.19: Overview of The RTR System.

6.9 System Overview

Figure 6.19 depicts the snapshot of the system. IPs on the VCI bus can be divided into two categories. Permanent members on the VCI bus have a static address, and dynamic IPs are allocated a VCI address during configuration. Thus each IP has to be developed with its associated VCI interface and wrapper, and they contain empty address registers which is programmed by the HOS before they are placed and routed in the RGA.

Among the permanent members on the VCI bus, the existence of several RAMs is important, since in this system several hardware blocks will be executing in parallel, they should not be slowed down due to multiplexing and sequential access to memory. The presence of an hardware Placer and Router is essential for rapid reconfiguration. They have the same functionality as a standard FPGA placer/router tool. The other essential physical ports of the system are permanent members of the VCI bus. The Bus Control Unit (BCU) is shown as a permanent member since it has to be present all the time. However this can also be a dynamic IP in case one wants to use protocols other than VCI, or different flavors of VCI.The choice of VCI as the interface protocol is largely influenced by the possibility of design reuse, since it is a well known standard.

6.10 Hardware Operating System

The HOS itself is a dynamic IP in the RTR system.

Each IP is stored in the RAM as a netlist (unplaced and unrouted) equivalent to `.VQM` (Altera) or `.NET` (VPR) format. The RGA status and address table stores the address of each dynamic IP and the CTs (Configurable tiles) occupied by it. Figure 6.20 depicts the

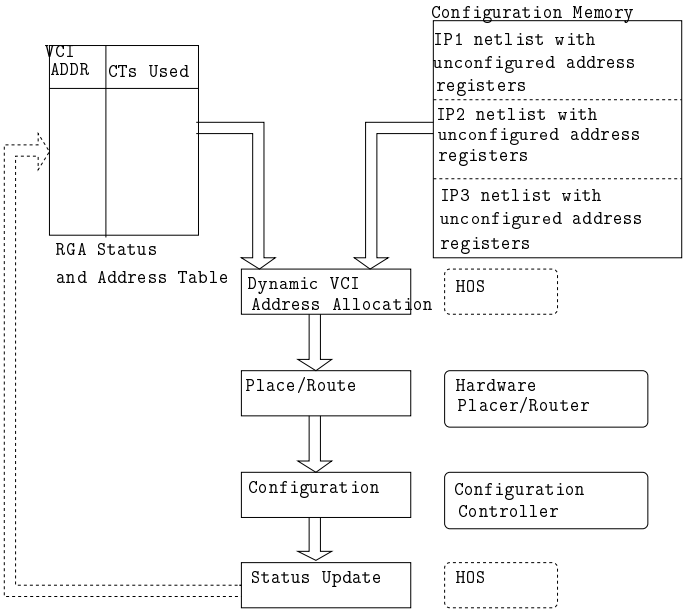


Figure 6.20: Basic Configuration Fetch

basic configuration fetch sequence. When an IP has to be executed, the HOS reprograms the address registers of the IP with an unallocated address and passes on this netlist to placer/router which is a permanent member on the VCI bus. It also passes the present status of the RGA to placer/router so that the new IP is not placed in the already occupied zone. Once placed/routed the HOS makes a request to the configuration controllers (permanent members on the VCI bus) to configure the IP, and initialize it. It then updates the RGA status.

Figure 6.21 depicts the basic interrupt handling sequence managed by the HOS. When it receives an interrupt, first the HOS looks for available space. If no space is available in the RGA (i.e. there are not enough unoccupied CTs to implement the interrupt handling IP) it decides to save the state of a low priority IP (IP2 in figure 6.21) that liberates enough CTs for the new incoming interrupt handling IP. The state is saved as a unplaced/unrouted netlist with proper register values. Next the HOS erases the concerned CTs and reconfigures the new IP using configuration fetch sequence.

We propose to use Hardware placer/router for increased configuration speed. This might seem too costly in terms of silicon but storing placed/routed netlists as executables has its own disadvantages. For example the system receives a high priority interrupt, and decides to erase a hardware block (say IP1) to configure the interrupt handling hardware block. If at some later time FPGA resources are released by an IP, IP1 can be configured in its place since it is in wait state. This is not possible if we use a placed/routed netlist in the memory. In that case IP1 has to wait the end of Interrupt Handling Hardware block.

Hardware placer/routers are an active area of research. The increasing density of transistors permits us to implement complex algorithms in hardware as noted by [92]. [92] has shown that, a systolic placer implementing simulated annealing algorithm can achieve 50x improvement in speed compared to software placers, albeit with some loss in quality. [55] discusses parallelizing the maze router.

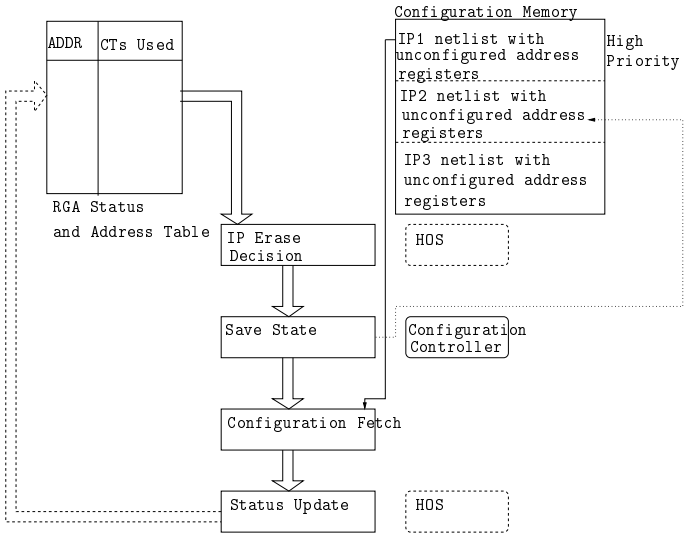


Figure 6.21: Basic Interrupt Handling.

6.10.1 Reconfiguration Speed

As can be seen in the previous experiment, the reconfiguration speed of RGA plays a major role in the overall efficiency of the device. To get more insight, and for comparison purpose we define two parameters related to the reconfiguration speed.

- **Effective Reconfiguration Speed** is defined as the number of basic reconfigurable cells(atoms) that can be reconfigured in one computing cycle. We also express the number of reconfigurable atoms as a percentage of the total atoms in the device. For example in our RGA the configuration cycle is same as the computing cycle. Each reconfigurable atom has 64 configuration bits. Thus effective reconfiguration speed of our RGA is 1/64 cells/computing cycle 0.024 %/computing Cycle.

- **Reconfiguration Ratio** for an IP is defined as the ratio of it's execution time Vs. it's reconfiguration time. A reconfiguration ratio less than one signifies that the IP spends more time in reconfiguration than in execution. During the reconfiguration the CLBS occupied by this IP is IDLE (Note that the RGA is not IDLE since other IPS are being run in parallel). In a real system IPs will be swapped in and out of the RGA over time. Hence a high reconfiguration ratio is desirable.

  For example in a microprocessor the instruction fetch cycle is 1 clock period and the execution cycle is at least 1 period. Hence the instructions has a reconfiguration ratio of 1.

From the above discussion we conclude that a high reconfiguration speed is essential. This can be achieved by parallel loading of bitstreams or highly optimizing the reconfiguration circuitry for speed.

6.11 Conclusion

FASE is a different approach towards implementing run-time reconfigurability in FPGAs. In FASE the soft IP cores are not constrained to fit in rows/columns, and there is no

necessity of explicit reset signals in the RTL design other than the global reset. However, such explicit initialization could also be done and, in that way, FASE is compatible with applications developed for existing RTR platforms.

Although this high granularity of reconfiguration may be used for better utilization of resources, it may also result in fragmentation, and increases the reconfiguration latency. Possible enhancements could solve the issue of de-fragmentation and reducing the reconfiguration latency.

A FASE planned improvement is to replace synchronous `CLB` by asynchronous ones in order to cancel the timing closure constraints. Moreover, with regard to security aspect, asynchronous logic coupled with RTR capability would greatly improve the robustness.

This paper presents a fully operational RTR FPGA(8x8) embedded in a SoC. The key features of this FPGA are that it is generated through an automated design flow, and it's capability to perform run-time reconfiguration free of electrical conflicts, which is demonstrated through an experiment. The advantages and disadvantages of the design flow automation are discussed and the results are compared with previous efforts of the same kind. We also present the user design flow which is very convenient, since it uses industry standard Altera synthesis tools along with VPR for place/route and bitstream generation.

Although commercial FPGAs like Virtex-5 offer RTR capability, the open nature of this project and access to the design flow allows the user to fine tune the FPGA according to his/her requirements(e.g in our case we added a functionality to selectively initialize each CLB). Our FPGA design flow is based on standard cell libraries, which permits the user to migrate easily to a newer technology, and experiment with various architectures. For the moment, this design flow is limited only to VPR-style architectures, Our future goal will be to enhance this flow to include various other type of architectures such as hierarchical FPGAs, and island style FPGAs with different tiling patterns (Octagonal, Hexagonal).

Chapter 7

An Asynchronous FPGA Architecture: SAFE

7.1 Introduction

While synchronous circuits design has reach a high level of performance, the clock distribution issues have become a real problem to cope with, therefore asynchronous circuits are more and more used in order to remove this clock-tree as well as dealing with the power consumption overhead which drastically increases with frequency. Moreover, the asynchronous circuits appear to be an interesting alternative to their synchronous counterparts for implementing cryptosystems [19][20]. In fact these latter are very sensitive to the so-called Side-Channel Attacks (SCAs) which aim at illegally retrieving secret information contained in cryptographic systems. At the same time programmable circuits have proved their validation role in the logic design flow and their high level of flexibility and performance. Therefore combining asynchronous logic with programmable circuits will be very helpful to explore and experiment the ability of asynchronous circuits, with or without specific countermeasures, to resist against power attacks and Fault Attacks.

7.2 Modelling The Side Channel

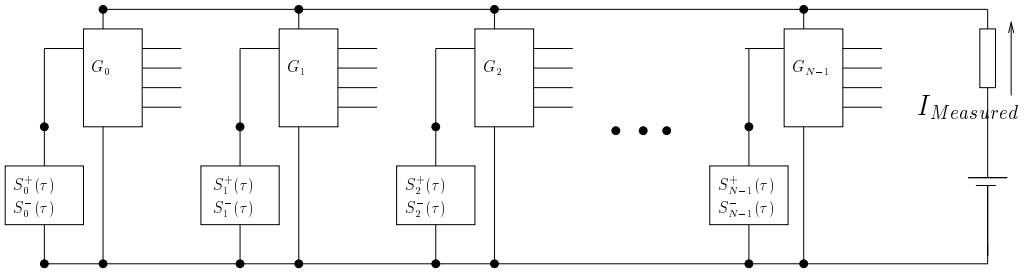
7.2.1 Dynamic Power Consumption Model

Static leakage power in CMOS doesn't contain any information about the computation being performed and is a constant hence we don't take it into account for Side-Channel Analysis. Power consumption is proportional to the current charged and discharged from the power supply.

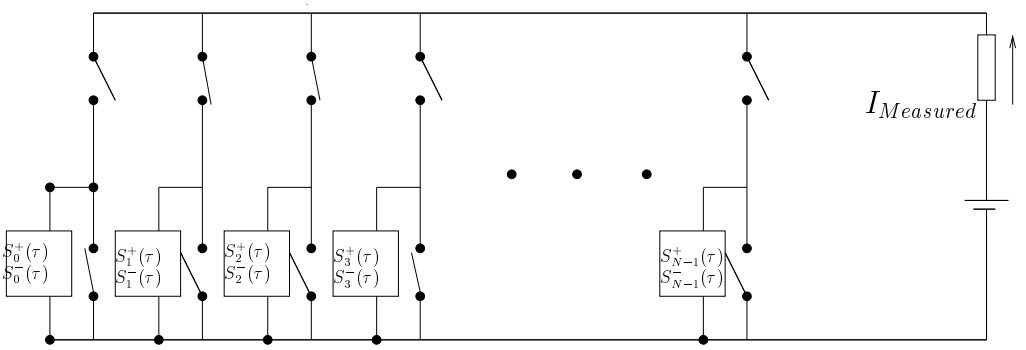
We model dynamic power consumption in CMOS in two levels.

- First the internal power consumption of the gate, which is due to charging and discharging of internal nets and transistor short-circuit currents inside the gate.
- Secondly the power consumption of the net driven by the gate which also includes the input capacitances of the driven gates.

Side-Channel Information is in the dynamic current profile of the circuit, thus we need a detailed model for this consumption. For this reason, each gate and each net is



(a) Based on Positive & Negative Step Response



(b) Model for a Buffered Net

Figure 7.1: Dynamic Power consumption model

associated with it's step current response(i.e the contribution of the component to the current $I_{measured}$(see fig. 7.1(a))).

Gate Level We consider gates with N inputs. Thus the gate is characterized by it's step current response as the input vector undergoes a transition $i_j \rightarrow i_k$ while the gate output is open.

$$S^{i_j \rightarrow i_k}(\tau) = I_{measured}(\tau) \tag{7.1}$$

and the gate is characterized by the set of all step responses corresponding to each transition.

$$\bigcup_{\substack{0<i<2^N-1 \\ 0<j<2^N-1}} S^{i_j \rightarrow i_k}(\tau)$$

Net Level Each net has only one input, hence it is characterized by it's positive and negative step response.

$$\begin{aligned} S^+(\tau) &= I_{measured}(\tau) \\ S^-(\tau) &= I_{measured}(\tau) \end{aligned}$$

while the input to the net is a positive or negative step.

We consider both positive and negative step response because the charging and discharging network for the net could be different in the actual layout.

We model a buffered net as depicted in figure 7.1(b). It is a delayed sum of the step responses of each segment, and the step responses for active gates(buffers). This point to

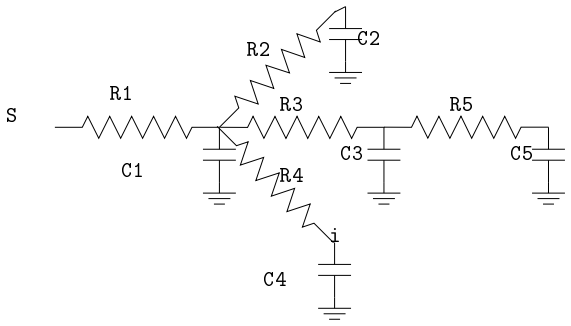


Figure 7.2: Delay model for the FPGA interconnect.

point delay is calculated using widely used Elmore Delay model as described in the next section.

7.2.1.1 Delay Model

To calculate the point to point delay in the above model we use the widely used Elmore delay model [133].

The Elmore delay is given by

$$\tau_i = \sum_{k=1}^{N} C_k R_{ik} \tag{7.2}$$

Where N is the No. of capacitance in the equivalent network and R_{ik} is given by

$$R_{ik} = \sum R_j \Rightarrow (R_j \in [\text{path}\,(S \rightarrow i) \cap \text{path}\,(S \rightarrow k)]) \tag{7.3}$$

7.2.2 Secure Place-Route Objectives

7.2.2.1 Indiscernability in power consumption

Gate Level In section 7.2 we have modeled the power consumption profile of a gate (A LUT in this case) as a set of step current response for each transition. Asynchronous logic gates are mapped into two symmetric LUTs as shown in figure 7.5. According to the 1-out-of-2 4-phase protocol, only one of these two gates will evaluate for each evaluate and precharge cycle. To guarantee that the current consumption profile of these two gates are similar, we try to assure that

- for a LUT each path from input to the output is indiscernible from each other.
- each path from the configuration memory point to the output is also indiscernible from each other.

If these conditions are met, it is not possible to predict which gate has evaluated, even if the corresponding dual-rails doesn't use the same inputs of the corresponding LUTs.

Net Level Figure 7.3 shows the effect of imbalance in capacitance and delay in dual rails, on side channel leakage. Even these small differences could be exploited for cryptanalysis [47]. In section 7.2 we have modeled each interconnect by it's positive and negative step current response. Ideally

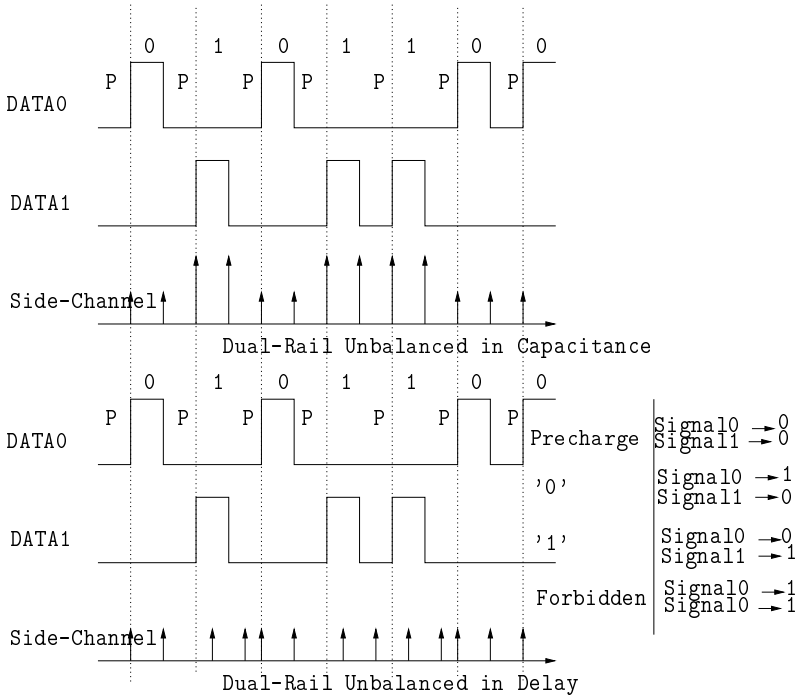


Figure 7.3: Imbalance in Capacitance and Delay Leaks Information

- The +ve and -ve step current response for each dual rail should be identical.
- For a buffered net, the buffers used should be identical, and each segment between buffers should also have identical step current response.

As a measure of indisceranbility, we use the cross-correlation of the step responses of each net. The higher the cross-correlation, the more difficult it is to predict which net has undergone transitions.

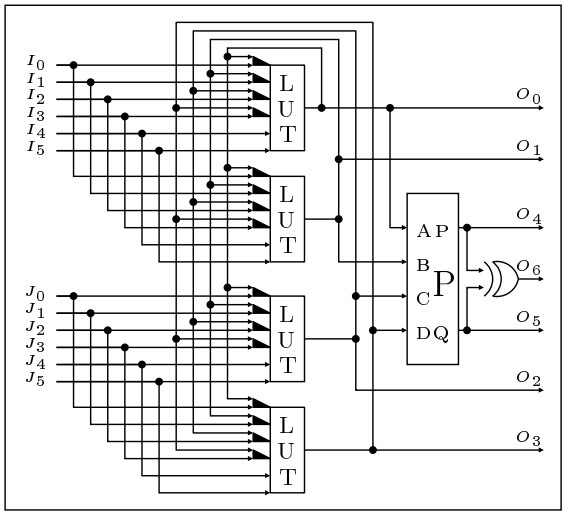
However to simplify the design procedure we make the following assumptions: the same length of wire of same width, charging the same capacitances, has a similar step current response, that is consumes the same current and causes the same delay irrespective of any bends.

In this respect we also define equitemporal lines for n-wire signals An equitemporal line (-···-·-) is the set of points attainable simultaneously by signals originating from synchronized sources (*i.e.* wave fronts.)

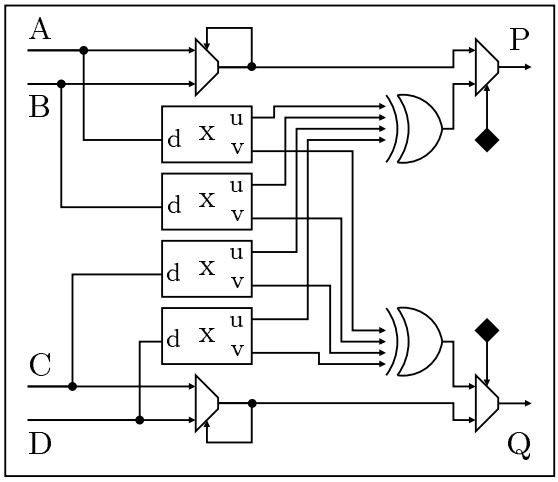
7.2.2.2 Indiscernability in EM emission

The radiation pattern measured at any point in space should be the same for each wire of a n-wire bus. For example, given a set of parallel wires (not twisted), we can choose a point closer to one of the wire and further from others. At that very point in space, radiation patterns emitted from different wires will be distinguishable.

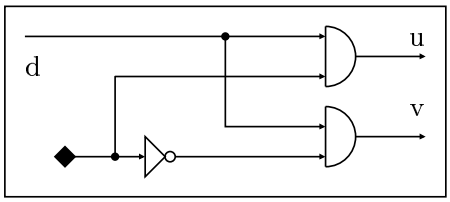
At the Gate level, the dual gates should be placed as close as possible, and at the net level we propose to route the dual rail signals as a twisted pair to deter EMA.



(a) PLB Functional Overview



(b) Block P in Detail



(c) Block X in Detail

Figure 7.4: PLB Overview

7.3 Logic Block Architecture

Fig. 7.4(a) depicts the structure of the PLB(Programmable Logic Block) able to handle the main QDI asynchronous styles. Black triangles at the left of LUTs are multiplexers controlled by programming points. The P output block is described more precisely by Fig. 7.4(b) and Fig. 7.4(c), in which the black diamonds represent programming points. The feedbacks from output to inputs are used to obtain the memory effect.

Details of the PLB and mappings can be found in [143]. A summary of all the implementable styles in the proposed PLB is given in the following table:

	A	B	C
2-phase EDGE	1	n.a.	n.a.
2-phase LEDR	0.5	1	n.a.
4-phase	0.5	1	2

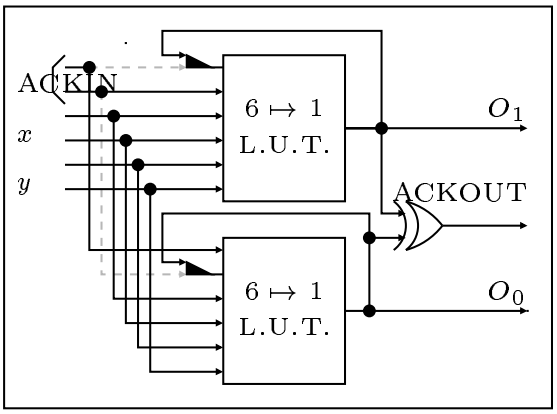


Figure 7.5: Mapping of 1-out-of-2, 2-Input Gates onto the PLB

The results are given in number of PLBs. The three configurations considered in this table are:
A: Dual Rail, 2-input gate with Acknowledge input,
B: Dual Rail, 3-input gate with Acknowledge input,
C: Triple Rail, 2-input gate with Acknowledge input.

For the purpose of this manuscript, we explain Mapping of 1-out-of-2, 2-Input Gates onto the PLB here. Other mappings can be found in [143].

7.3.0.3 1-out-of-2, 2-Input Gates

Let $f(x,y) : \mathbb{F}_2 \times \mathbb{F}_2 \mapsto \mathbb{F}_2$ a two-variable Boolean function. The inputs are represented by 4 wires: x_0, x_1, y_0 and y_1, to which a synchronization signal (*Acknowledge*), $\mathcal{S}_{in}$, is added and the output signal O represented by two wires: O_0 and O_1, together with an acknowledge output $\mathcal{S}_{out}$. The individual values of these wires are functions of x and y, respectively denoted $f^0(x,y)$ and $f^1(x,y)$.

The equations of the outputs are:

$$\begin{aligned} O_1 &= \begin{cases} f^1(x,y) & \text{if } x,y \neq \mathbf{\Omega} \wedge \mathcal{S}_{in} = 0\,, \\ 0 & \text{if } x,y = \mathbf{\Omega} \wedge \mathcal{S}_{in} = 1\,, \\ O_1 & \text{otherwise}\,, \end{cases} \\ O_0 &= \begin{cases} f^0(x,y) & \text{if } x,y \neq \mathbf{\Omega} \wedge \mathcal{S}_{in} = 0\,, \\ 0 & \text{if } x,y = \mathbf{\Omega} \wedge \mathcal{S}_{in} = 1\,, \\ O_0 & \text{otherwise}\,, \end{cases} \end{aligned} \tag{7.4}$$

in which "$x,y = \mathbf{\Omega}$" stands for "$x = \mathbf{\Omega} \wedge y = \mathbf{\Omega}$".

Eq. (7.4) shows that O_0 and O_1 are functions of 6 Boolean variables. Thus the minimal practical size for the LUTs is 64 bits, which can implement a 6-bit $\mapsto$ 1-bit function. As there are two output bits the minimal size of the PLB is 2 LUTs. The $\mathcal{S}_{out}$ output can be computed as $(O_1 \vee O_0)$. However, for homogeneity with the 2-phase protocols, we use $(O_1 \oplus O_0)$ instead. Note that, as the $O_0 = O_1 = 1$ state is forbidden, the two functions are identical on the allowed domain.

A feedback is necessary to obtain the memory effect. One of the inputs to the LUTs must thus be programmable between the input to the PLB and the feedback. As the inputs

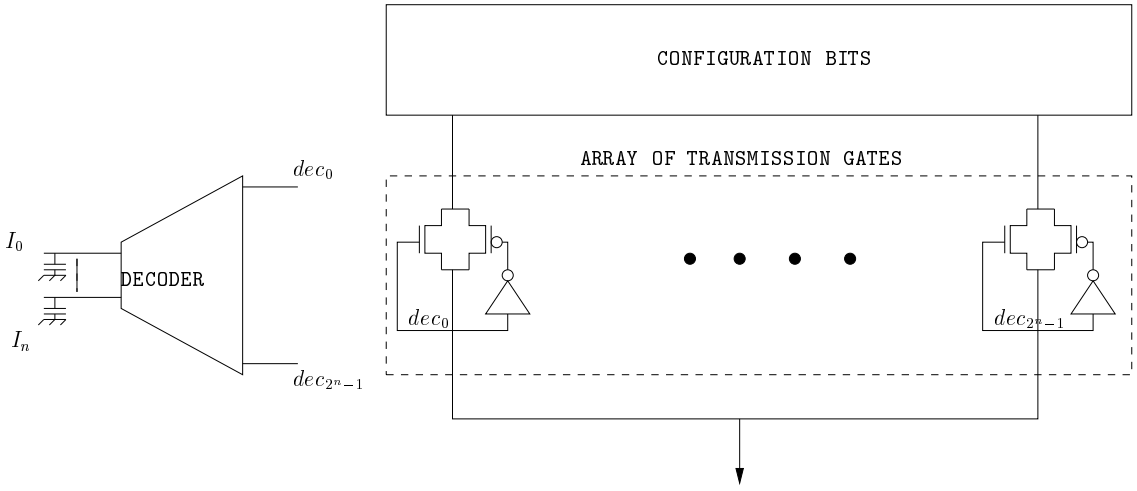


Figure 7.6: LUT implementation

to the LUTs are the same, with the exception of the feedback wires, there can be a single connexion box to the routing network. Fig. 7.5 shows the minimal structure of the PLB, which allows to implement 2-input gates with synchronization.

The associated wiring is:

$$\begin{aligned} O_0 &= \text{LUT6}(\, O_0 \,, \mathcal{S}_{in} \,, x_1 \,, x_0 \,, y_1 \,, y_0 \,)\,, \\ O_1 &= \text{LUT6}(\, \mathcal{S}_{in} \,, O_1 \,, x_1 \,, x_0 \,, y_1 \,, y_0 \,)\,. \end{aligned} \tag{7.5}$$

In Eq. (7.5), each wires of x and y is loaded with exactly the same number of inputs. Note that the $\mathcal{S}_{in}$ signal is connected twice to the routing network.

7.3.0.4 Balanced LUT Implementation

Figure 7.6 depicts the LUT implementation scheme to achieve the objectives set in section 7.2.2. In a classical LUT implementation with multiplexer trees each input is loaded with a different capacitance, and also the logic depth from configuration bits to the outputs vary with the input activity.

To circumvent these drawbacks, all input capacitances have been balanced by correctly adjusting the sizes of decoder's inverter, and a unique logical depth is imposed between the configuration bits and the LUT output. More details about the LUT implementation and simulation results can be found in [135].

Figure 7.7 describes the actual PLB layout. The four $6 \rightarrow 1$ LUTs are placed symmetrically, and in between the input multiplexers are placed which connects the LUT inputs to the routing resources, and feedbacks. The block P is place at the top. All the twelve inputs are placed at the top, and the seven outputs at the right of the PLB layout.

7.4 Routing Architecture

We use the traditional Mesh routing architecture and associated nomenclature as explained in [54].

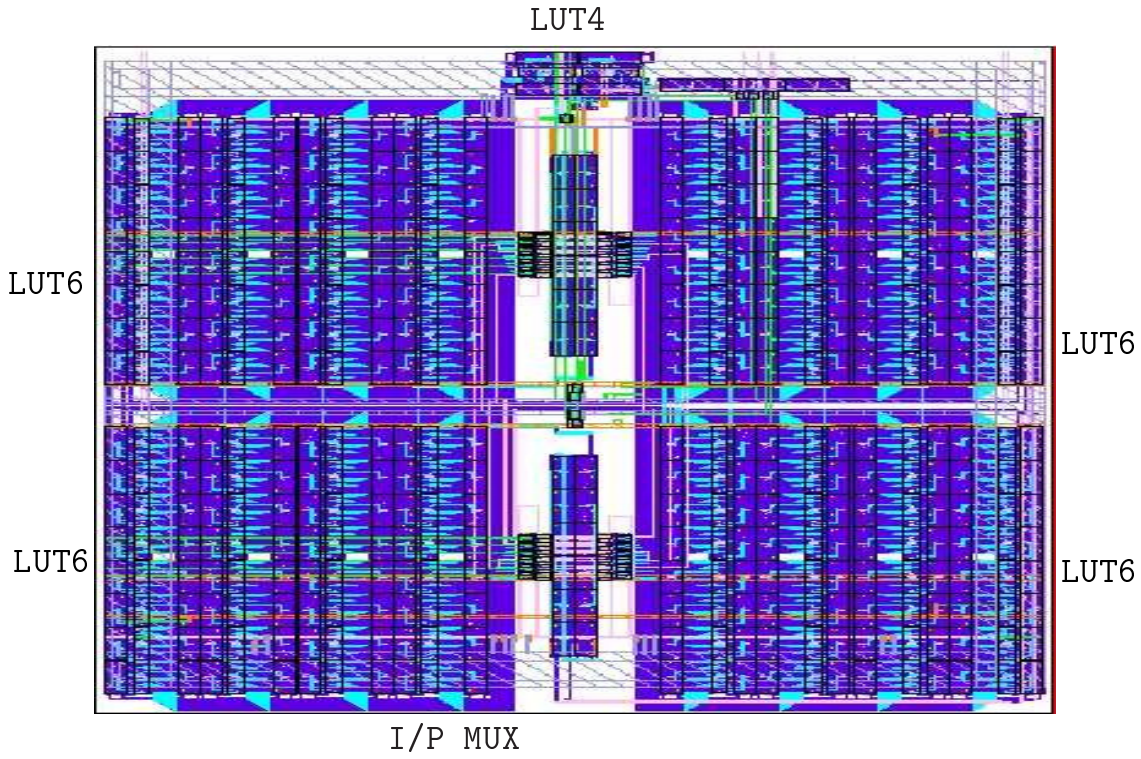


Figure 7.7: PLB Layout

7.4.1 Subset Routing Architecture

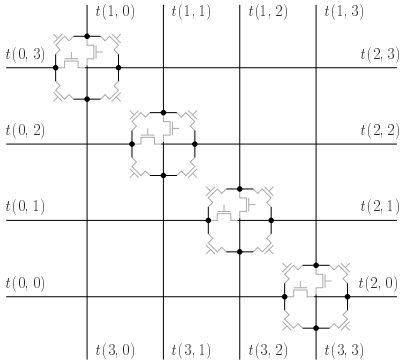
A subset switchbox [54] can be built by repeating a basic six-way switch-points along a diagonal, as shown in figure 7.8(a). We consider that the diagonal formed by the six-way switch-points makes up equitemporal signals(see section 7.2.2) if these signals are outputs of the same FPGA logic element CLB. Figure 7.8(b) shows the routing matrix using a subset switchbox. Connection boxes from the equitemporal lines to the CLB inputs/outputs are considered as equitemporals. They are discussed in section 7.4.5. In figure 7.8(b), the dual pair signals corresponding to connections {A, A'} and {D, D'} have exactly the same length and the same electrical characteristics. The same goes for buses {B, B'} and {C, C'}. Notice that the dual-rail signals are not necessarily routed in an adjacent way (case of A and D) and that it is possible to route in the same fashion multi-wire signals.

7.4.2 Twisted Pair Routing Architecture

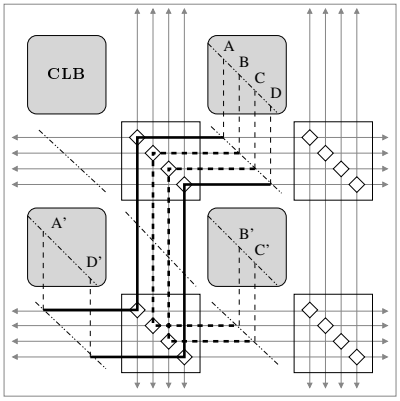
As a countermeasure against information leakage through EM radiations, we propose to route every n-rail signal as a twisted bus. Figure 7.9(a) shows the advantages of using a twisted pair compared to parallel routed wires. If we consider the twisted pair as made up of several elementary radiating loops, we see that the radiation from a loop is canceled by that of adjacent loops.

In addition to reducing EM compromising radiations (outputs), the twisted bus gains immunity from its EM vicinity (inputs). Consequently, twisting signals bundles reduces cross-talk,

In order to route any n-rail signal as a twisted bus throughout the FPGA, two novel

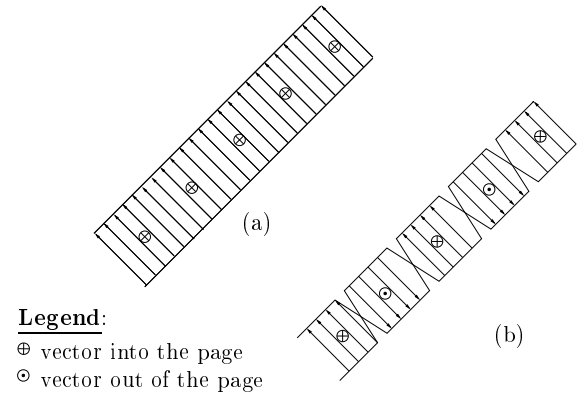


(a) *Subset* switchbox using six-way switch-points.

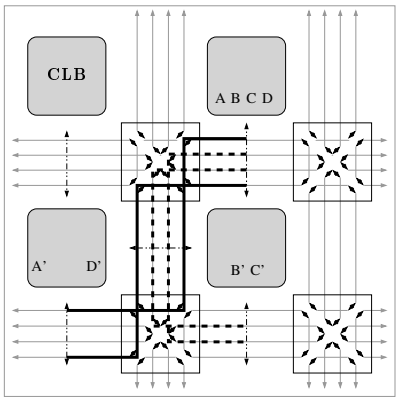


(b) Equitemporal lines for *subset* switchbox routing.

Figure 7.8: *subset* switchbox.



(a) Electric & magnetic fields orientation in an untwisted (a) and in a twisted (b) pair.



(b) Equitemporal lines for the *twisted-pair* switchbox.

Figure 7.9: *twisted-pair* switchbox.

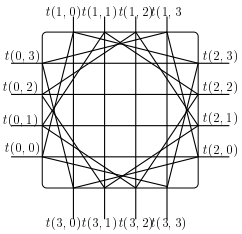
switchboxes are introduced.

7.4.3 Twist-on-Turn Switch Matrix

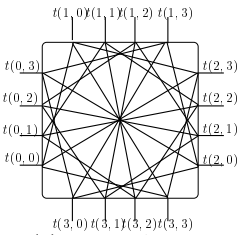
The basic idea behind this switchbox is that every pair or n-uplets of signals deflected by the switchbox must come out twisted. As shown in figure 7.9(b), every $\pm\pi/2$ bend through this switchbox is a twisted pair. We can express this switchbox using the notation described in [91] as:

$$\mathbf{S} = \bigcup_{i=0}^{W-1} \left\{ \begin{array}{c} [t(0,i), t(2,i)]\,, \\ [t(1,i), t(3,i)]\,, \\ [t(0,i), t(1,i)]\,, \\ [t(1,i), t(2, W-i-1)]\,, \\ [t(2,i), t(3,i)]\,, \\ [t(3,i), t(0, W-i-1)]\,. \end{array} \right\}$$

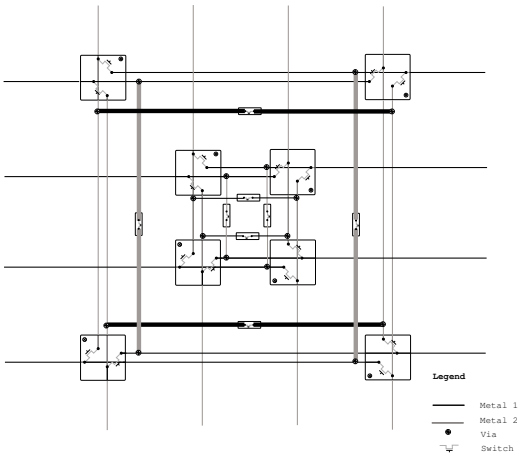
each terminal is represented as $t(j,i)$, where j denotes each subset corresponding to each side ($0 =$ left, $1 =$ top, $2 =$ right, $3 =$ bottom) and $i \in [0, W[$ denotes the position of the terminal in that subset. Connection boxes from the equitemporal lines to the CLB inputs/outputs are considered as being equitemporal perpendicular to the routing channel. They are discussed in section 7.4.5. In figure 7.9(b), the dual pair signals corresponding



(a) Twist-on-Turn.



(c) Twist-Always.



(b) Twist-always switch matrix layout scheme.

Figure 7.10: The twisted-pair switchboxes.

to connections {A, A'} and {D, D'} have exactly the same length even if they cross at the switching box. It is exactly the same for buses {B, B'} and {C, C'}.

When turning, this switch matrix introduces a small imbalance for the arrival time on the deflecting switch point. If the switch point is implemented with *passive* gates, this balance violation is not observable by an attacker. The counterpart is that the channels must be buffered, which can safely be done with *active* gates, because every wire in a channel is equitemporal.

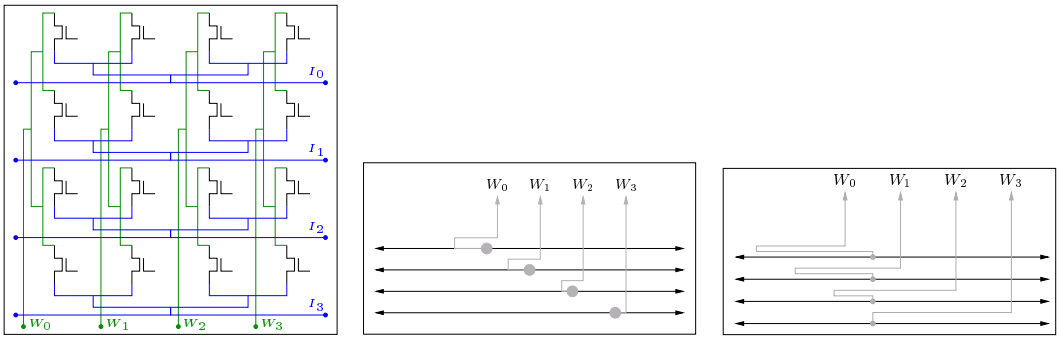
7.4.4 Twist-Always Switch Matrix

The twist-on-turn matrix does not twist buses when they are routed straight. This matrix can be transformed in a twist-always matrix by twisting the wire i wire with wire $W-1-i$ for straight connections, as shown in figure 7.10, W being the number of channels.

This matrix allows the use any 1-of-n (asynchronous) style, as it is possible to twist a number of lines greater than two.

This switchbox cannot be implemented with traditional six-way switch-points, even if the number of transistors remains the same. A possible implementation of the twist-always switch box is shown in figure 7.10(b). It can be laid out in silicon with two interconnect layers and by repeating two basic patterns over space. Note that for straight (*e.g.* from left to right) connections, the outer rails are drawn wider than the inner rails to compensate for the difference in lengths. Alternatively, every wire can keep the same nominal width, but inner rails are forced to zigzag so as to make up for their shorter length. For bends, every rail traverses an equal distance, hence this compensation is not required.

These new switchboxes are close to conventional universal/subset switchbox in terms of connectivity. Hence we can expect similar performance in routability of netlists in the FPGA.



(a) Balanced crossbar for the connection box.

(b) For Subset

(c) For T-pair

Figure 7.11: Balanced Crossbar and Connections between channel wires and crossbar.

7.4.5 Connection Box Implementation

7.4.6 Cross-Bar Connection Box

As depicted in figures 7.8(b) and 7.9(b), a signal routed from one equitemporal line to another have the same delay. Therefore the connection box (C-Box) between the W channel wires and the CLB $I \in [0, W[$ inputs/outputs should also keep this equitemporality. We propose to use a crossbar connection box based on balanced binary trees, built according to the following three rules: *(i)* from the channel, W trees have I equal-length branches, *(ii)* from the CLB, I trees have W equal-length branches, *(iii)* the two trees are superimposed orthogonally and the $W \times I$ branches from each tree type meet via a switch point.

Figure 7.11(a) illustrates the layout of the balanced crossbar with $W = 4$ and $I = 4$, using only two metal layers (represented with two different thicknesses.) The crossbar area is $W \cdot \lceil \log_2(I) \rceil \times I \cdot \lceil \log_2(W) \rceil$ square routing pitches, and can be freely depopulated without altering its security level.

7.4.7 C-Box for Subset & Twisted-Pair Switch Matrix

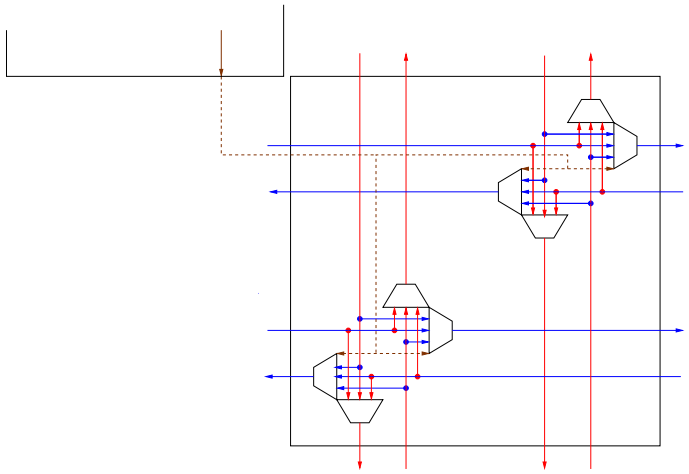
The equitemporal lines are either diagonal (for the subset switch matrix, *cf* Fig. 7.8(b)) or horizontal/vertical (for the twisted switch matrix, *cf* Fig. 7.10(b).) The connections between the channel and the crossbar should compensate the wire length delays. A solution for both cases is illustrated in figure 7.11.

Example layouts and statistics of the T-pair switchbox, and the binary-tree connection box can be found in [141].

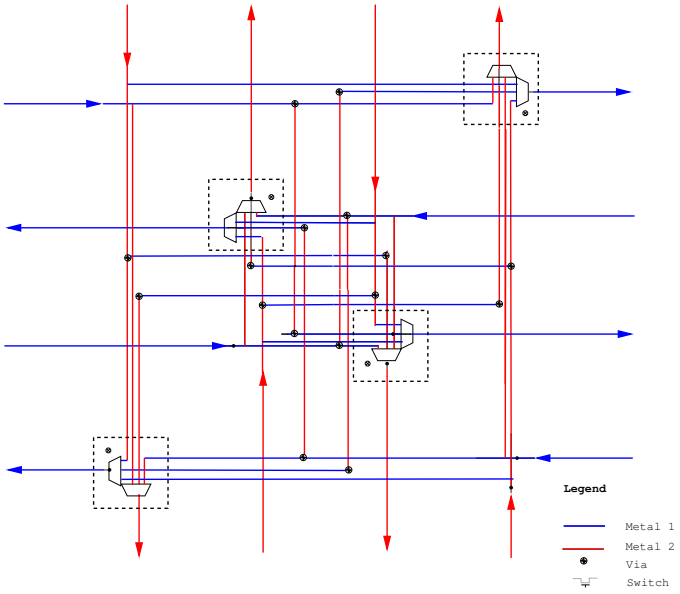
7.5 Single Driver Architecture

Single driver segments are shown to give better delay performances and better area-delay product for an FPGA [76, 78]. These benefits are result of less loading of interconnect segments, and availability of equal no of tracks in each direction as in traditional bidir-tri routing architecture.

Figures 7.12(a) shows how the subset switchbox layout scheme 7.8(a) can be ported to single driver architecture. Note that the connection box nets are routed as a binary tree in both X and Y direction. Figure 7.12(b) shows the layout scheme for the T-pair switchbox



(a) Single Driver Subset



(b) Single Driver Tpair

Figure 7.12: Porting the solution to Single Driver Architecture

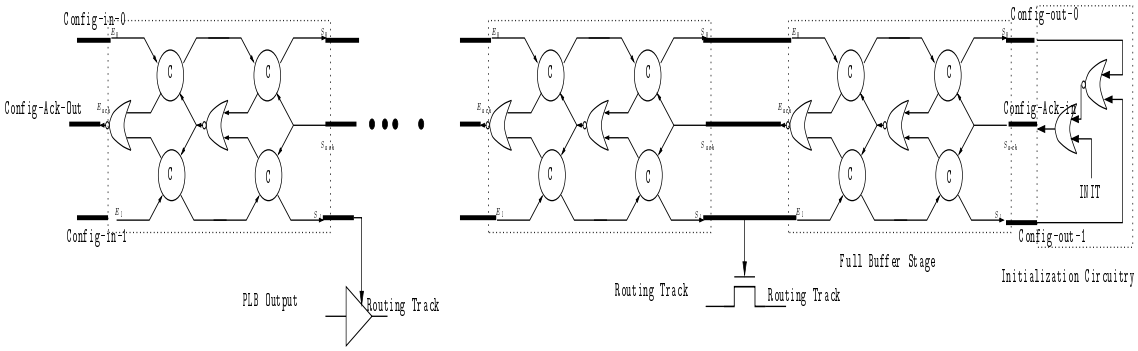


Figure 7.13: The Asynchronous Configuration Chain with Reset Circuitry

with single driver interconnects. Note that this scheme too uses a basic switch-point as in 7.10(b) which is rotated as required.

7.6 Configuration Architecture

Figure 7.13 describes the configuration architecture for our asynchronous FPGA "SAFE". The configuration chain is designed assuming 1-out-of-2 4-phase protocol (as described in 3.3.2). It consists of a series of Full Buffers(figure 3.6(b)) terminated by a `initialization circuitry`. The function of the `initialization circuitry` is to bring the whole chain to $('0','0')$ state, and to avoid any invalid state $('1','1')$ after power up, or to erase a previous configuration. For *initialization* the signal `INIT` is put to '0', and `config-in-0` ,`config-in-1` is held at '0'. The '0's from the configuration input will propagate throughout the chain making a *reset* action. For any invalid state $('1','1')$ after power up, we can see that the output of the *nor* gates in the chain will be '0', so the invalid state will be overwritten by $('0','0')$ from the input of the chain. Since the signal `INIT` is held at '0' during the initialization phase, the acknowledge input to the last stage is '0' for

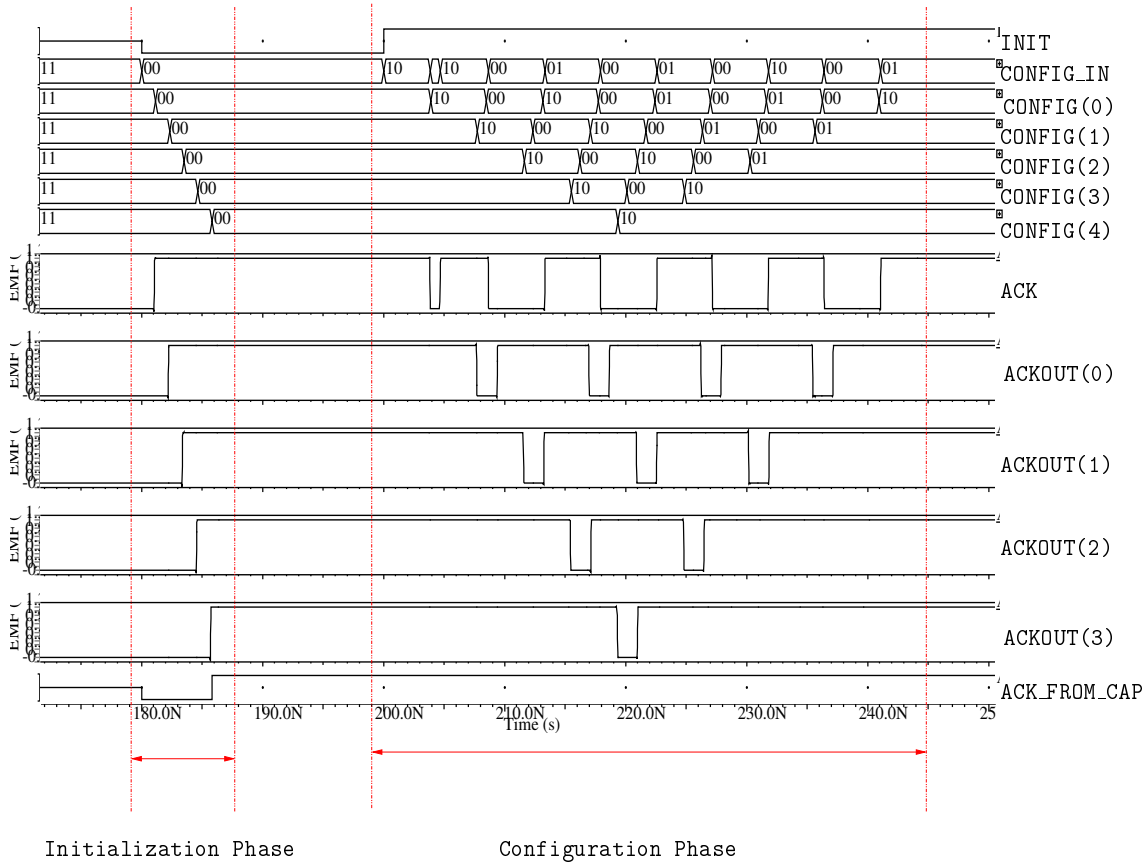
$$(\texttt{config-out-0}, \texttt{config-out-1}) = (0,0) or (0,1) or (1,0).$$

It will only accept a $(0,0)$ from the previous stage.

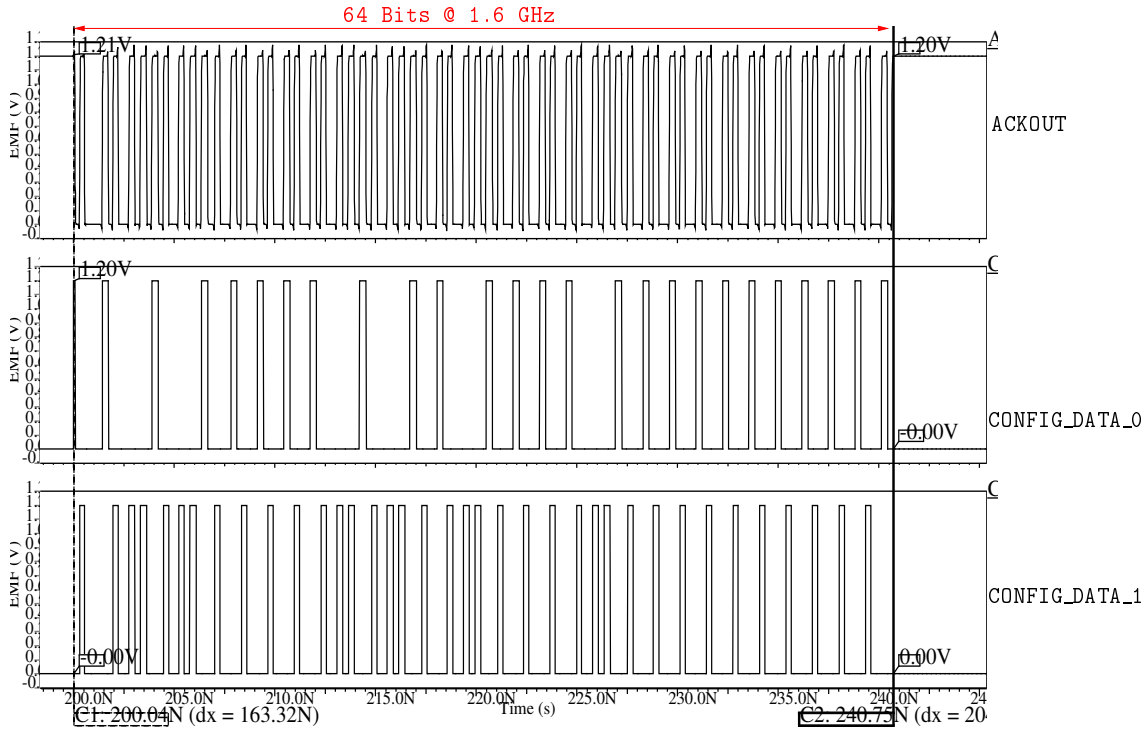
During normal operation, `INIT` and consequently the acknowledge input to the last stage is held at '1', hence chain will accept any valid state until the chain is full.

Figure. 7.14(a) shows the mixed-signal simulation results for an asynchronous configuration chain with 5 *full-buffer* stages and a initialization cap as explained in fig. 7.13. During the initialization phase, the input to the configuration chain `CONFIG_IN` and the `INIT` signal is forced to '0'. We can see that the "00" propagates along the configuration chain, and the acknowledge signals are initialized to '1' (READY) state. We can see that even if the configuration signals are powered up at an invalid ("11") state, the reset function of the *initialization cap* brings the chain to a valid *precharge* state which is ready to accept a new configuration.

In the configuration phase the `INIT` signal is kept at '1', and we can see in fig. 7.14(a) the configuration bits advancing through the chain. The waveforms presented are from a mixed-signal simulation, where the 5 *full-buffer*s are analog, the *initialization cap* and the testbench controlling `CONFIG_IN` are digital circuits. Because of this reason the signals `ACK` and `ACK_FORM_CAP` at two ends of the chain have different delays.

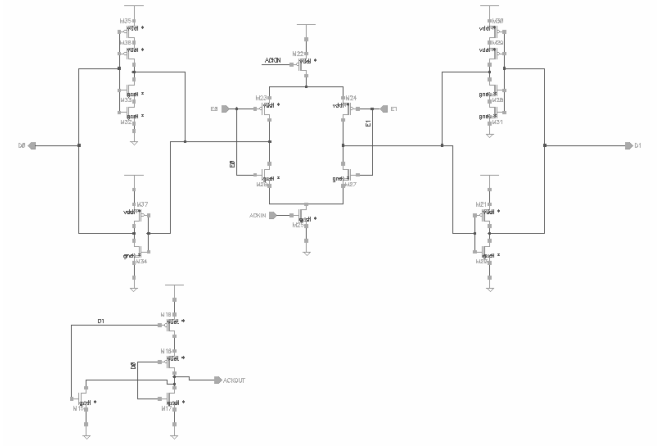


(a) The Initialization and Configuration Phases

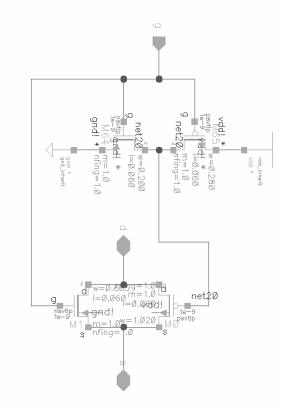


(b) The Maximum speed of the Configuration Chain (1.6GHz), Simulated with RC extracted view

Figure 7.14: Simulation of the Asynchronous Configuration Chain



(a) Half-Buffer Standard Cell



(b) Transmission Gate Standard Cell

Figure 7.15: Prototype: Schematic of Basic Standard Cells

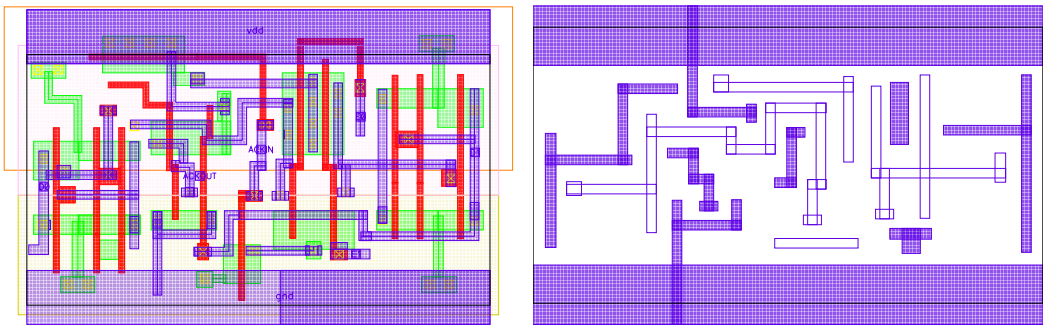
In Figure. 7.14(b) we present the simulation results of the configuration of a single 6-input LUT with parasitic RC to determine the highest achievable configuration speed. We used STARRCXT for parasitic extraction and ADVANCE MS(Mentor Graphics [119]) for simulation. In the waveforms the input is the digital input to the configuration chain and the acknowledgement(`ACK`) from the chain which is a analog circuit with parasitic RC. The rise-time(t_r), fall-time(t_f) and delay of the virtual analog to digital converters for simulation are kept very small($\sim$ 0.1 ps) so that they don't affect the simulation results.

From fig. 7.14(b) we can see that we can configure 64 bits in about $\sim 40ns$. Thus the maximum achievable configuration speed is around $\sim 1.6GHz$. This high speed is particular to the asynchronous configuration chain. This is mainly due to the fact that the each configuration stage output see very small capacitative load, since it has a fanout of 2: one for the next stage and one for the switch connected to the output(see fig. 7.13). Although this is also true for synchronous shift register chains, their speed is often limited by the clock tree skew. The reader might note that there is a small difference in the acknowledgement(`ACK`) delay for `CONFIG_DATA_0` and `CONFIG_DATA_1`. This is due to the fact that in our FPGA the switches are connected only to `CONFIG_DATA_1`, thus it has a bigger delay than `CONFIG_DATA_0`. This is also particular to the asynchronous configuration chain, because in the synchronous case we had to use the worst-case delay to design the clock tree.

7.7 Prototype

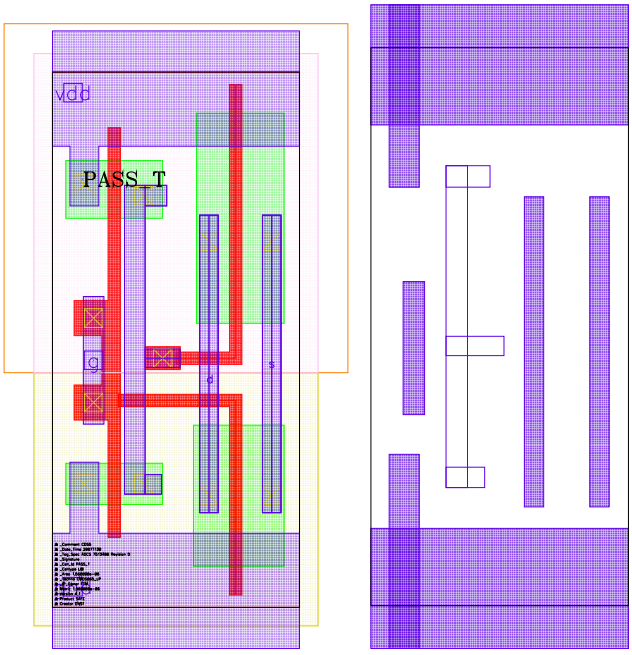
7.7.1 Basic Cells

Figure 7.15 and Figure 7.16 describes the schematics and layouts of basic standard cells in SAFE. The PLB is designed as a full custom block as shown in 7.7, whereas the interconnect portion of SAFE has been designed using the Standard Cell methodology with customized routing for balanced interconnect. For these reasons we laid out these basic cells similar to the standard cells used in ST Microelectronics Design Kit for CMOS065 process.



(a) Half-Buffer Standard Cell

(b) Abstract Seen By P/R Tool



(c) Transmission Gate Standard Cell

(d) Abstract Seen By P/R Tool

Figure 7.16: Prototype: layout and abstract of basic standard cells

7.7.2 Layout

Figure 7.17(a) shows the 3×3 prototype asynchronous FPGA. This chip is being fabricated using ST Microelectronics 65 nm 7-layer process called CMOS065. The FPGA channel width is 8, and there are 9 I/Os for each side. Figure 7.17(b) shows how the segments are routed inside the FPGA.

This FPGA have been laid out using a automatic flow as described in [137]. For balanced place and route:

- We first placed the switches, in a symmetric fashion as outlined in the layout schemes.
- All channel segments are manually routed to achieve required balance.
- Configuration memory points and signals are placed/routed automatically.

Since size of the FPGA mainly depends on size of switches and configuration memory points, and not limited by the routing area. Incorporation of balanced routing doesn't result in a overhead in terms of area.

The prototype occupies an area of 1111.6000μm $\times$947.6000 μm in silicon and contains approximately 200,000 transistors.

7.8 Experiments on Extracted Netlist

7.8.1 Experimental setup

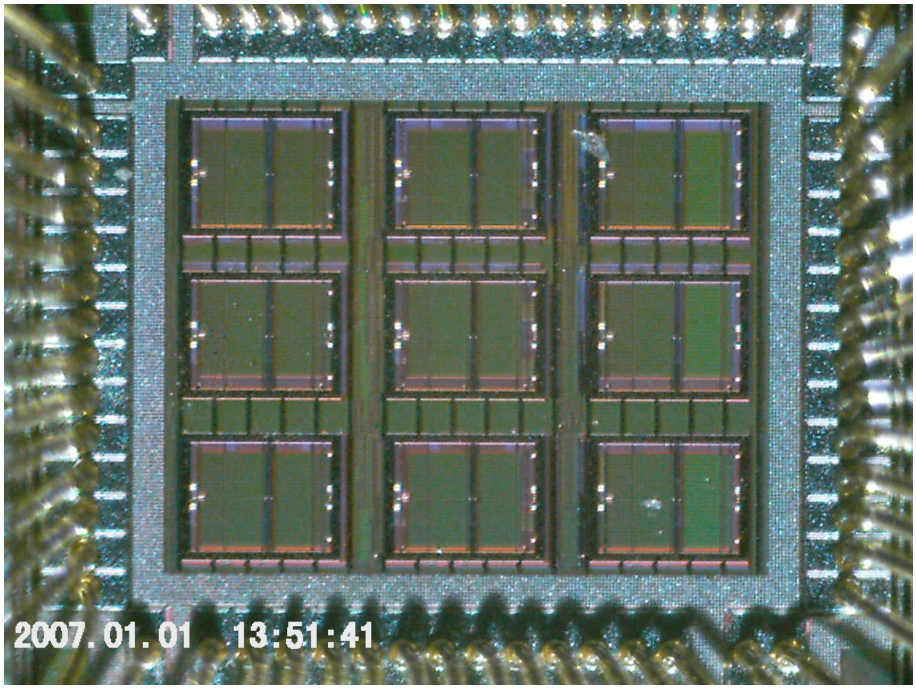
In this section we present the experiments carried out on the extracted FPGA layout. We used STARRCXT [121] for parasitic extraction and ELDO [119] for simulating one hop of the FPGA. Figure 7.18 describes the experimental setup used for this experiments. One hop consists of a path from the output through the right connection box, switchbox, the top connection box and to the input at the top.

7.8.2 Area-Delay Product

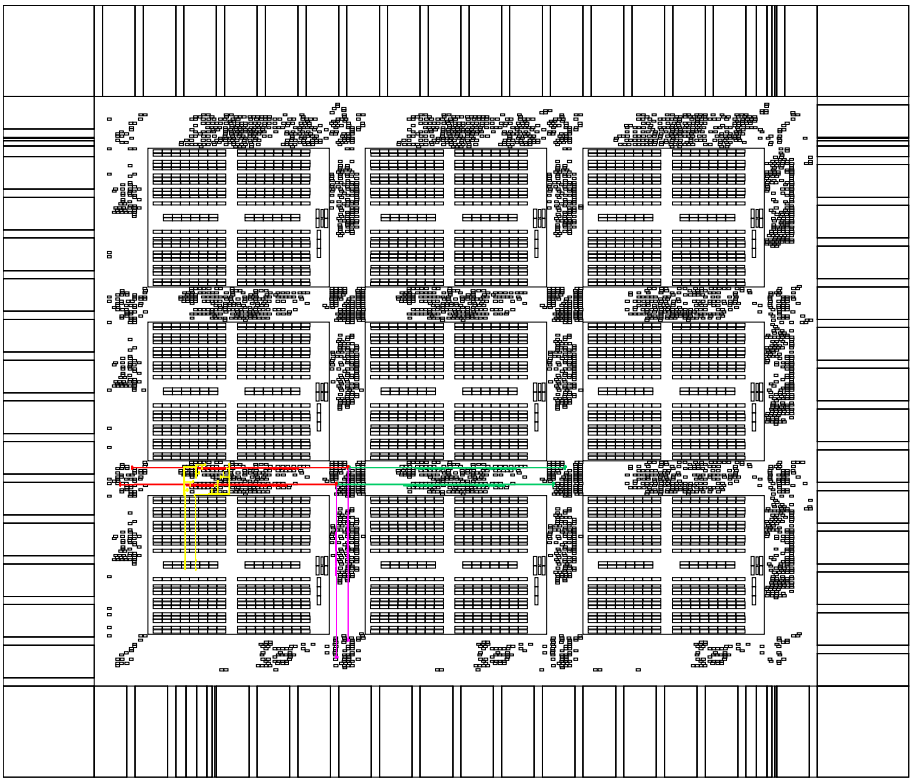
The goal of this experiment is to find out additional delay overhead due to the balanced routing constraints. A positive and negative step is applied to 7 1-hop paths (see fig 7.18), and the results are shown in figure 7.19. The average delay in the balanced case is 1045 ps whereas for the automatically routed segments the delay is 901 ps. This signifies a 16 percent increase in delay due to balanced routing. Keeping in mind this added delay, in a full-scale FPGA we might route only a few no. of tracks in a balanced manner (Secure Tracks) which will be used to route sensitive signals. This constraint should be handled by the FPGA CAD(place/route) tools.

7.8.3 Balancedness

A setup similar to figure 7.18 is used for this experiment. We simulate all paths from two outputs to all the 12 inputs, and we measure the step current response.(see fig 7.20(a)). We take each possible pair from this set and measure the cross-correlation between them. A higher value of cross-correlation means that the step current response for those paths are identical. Figure 7.20(b) shows the results.



(a) 3x3 Prototype Asynchronous FPGA (CMP Run S65C8_1)



(b) Routes inside the FPGA

Figure 7.17: Prototype

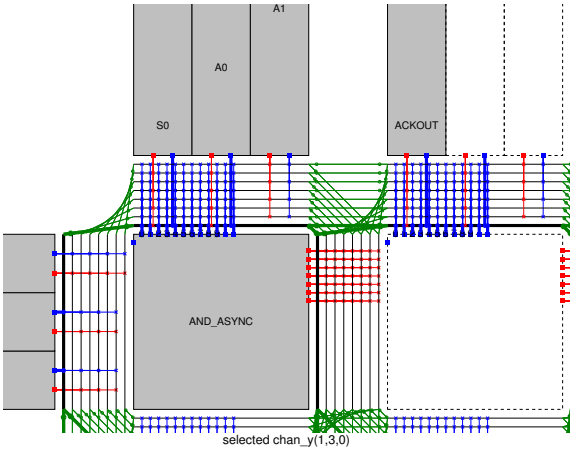


Figure 7.18: Experimental Setup

7.8.4 Hop Mismatch

In this experiment we try to find out the effect of hop-mismatch between the individual rails of a dual-rail signal. for this purpose one signal is simulated with 1-hop and the other with 2-hops using the same segment. Figure 7.21(a) shows the imbalance in delay and supply current for each rail. Figure 7.21(b) shows the cross-correlation between the step current responses of these two signals. We see a noticeable difference between a hop-mismatch of 0 and that of 1. To avoid this FPGA CAD(place/route) tools must guarantee the balanced routing of dual rails(i.e equal no. of hops).

7.9 Experiments on Silicon

The actual silicon measurements on FPGA SAFE has been carried out with the following test fixture(please see figure 8.5 in Appendix).

- ALTERA DE2 Board [67] which is used to provide input signals and and store output signals from SAFE. The ALTERA DE2 board is controlled from PC through a Python Shell on the PC side and DE2 USB API on the DE2 side.

- The FPGA SAFE itself is mounted on a PCB (in-house production) with buffers for level translation (3.3v to 1.2v). The SAFE IOs are all bidirectional, but to limit the use of buffers the signal lines on the PCB are directional. Figure 7.23 each group of three I/Os(per row and column) are grouped as one Output, one Input and one Input/Output(selectable with jumpers). This is done with 1-out-2 coding in mind for basic testing. By selecting the direction of the Input/Output in the group, the group can be used either as a asynchronous input(i.e 2 input signals and one output signal for acknowledge) or an asynchronous output(2 output signals and one input signal for acknowledge). The reader may note that this is just a constraint due to the use of buffers(level translators on the PCB), and in future it can be modified to be used with DE2 board I/Os directly, since SAFE I/Os can support 1.8v.

- VPR is used to generate the bitstream for interconnect routing, whereas PLB bitstreams are hand-coded for the moment.

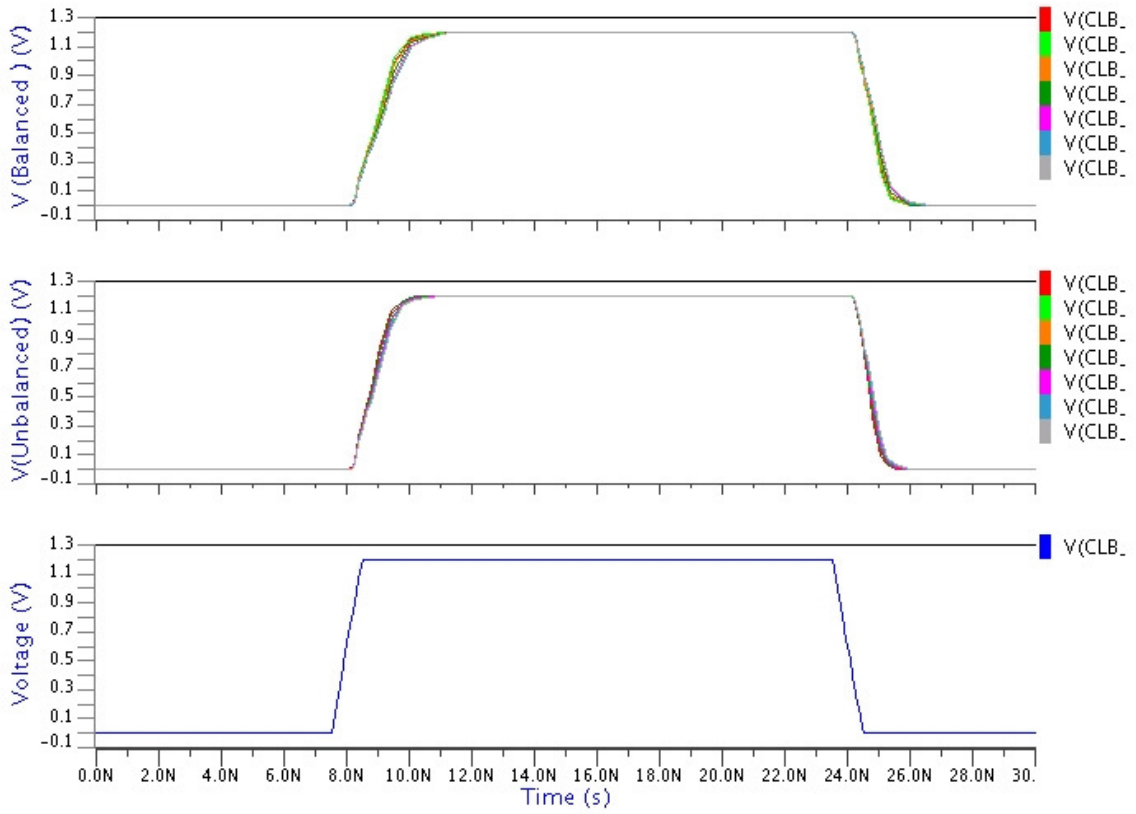


Figure 7.19: Delay Overhead

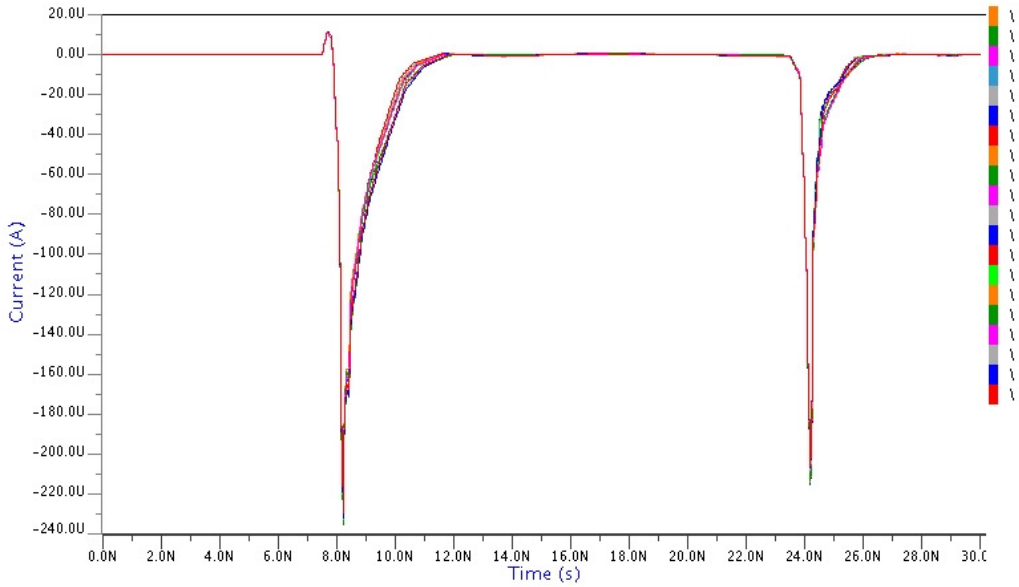
7.9.1 Experiment: Configuration

The sequence for this experiment is the following:

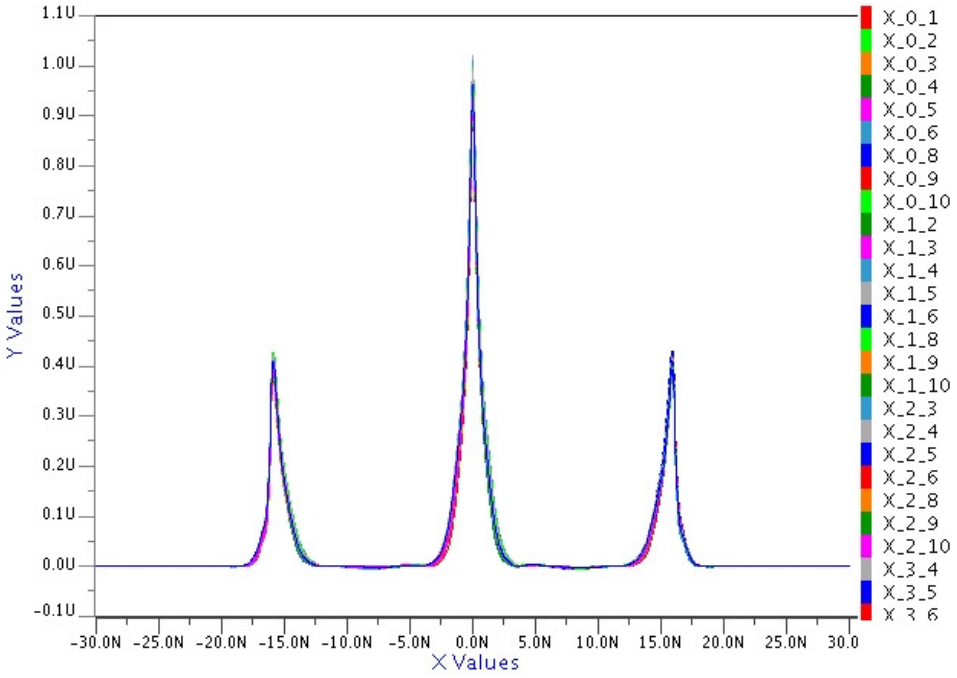
- First the asynchronous configuration chain inside SAFE is initialized, by putting the *INIT* input of SAFE to '0' and then released after some time.
- The bitstream file generated by VPR is loaded onto DE2 board RAM from PC. This bitstream is then converted to asynchronous 1-out-of-2 coding and sent to the *configuration-0* and *configuration-1* input signals of SAFE.
- The DE2 Board monitors the *acknowledge* output from SAFE, and puts a new value in the configuration chain following 4-phase Handshake. It also counts the Number of acknowledges received. For a successful configuration it should receive exactly 4692($x1254$) acknowledgments.

The results of this experiment are: When the bitstream contains only '0's the configuration is successful each time and with a very high speed. We tested upto $50MHz$, the DE2 board frequency. When the bitstream contains '1's the configuration only succeeds at a low speed(around $10KHz$)

Despite the fact that due to asynchronous coding '0's are '1's are equivalent in terms of transitions, we think that this behaviour might be due to a bug which we will be trying to explain with simulations in subsection 7.9.4.

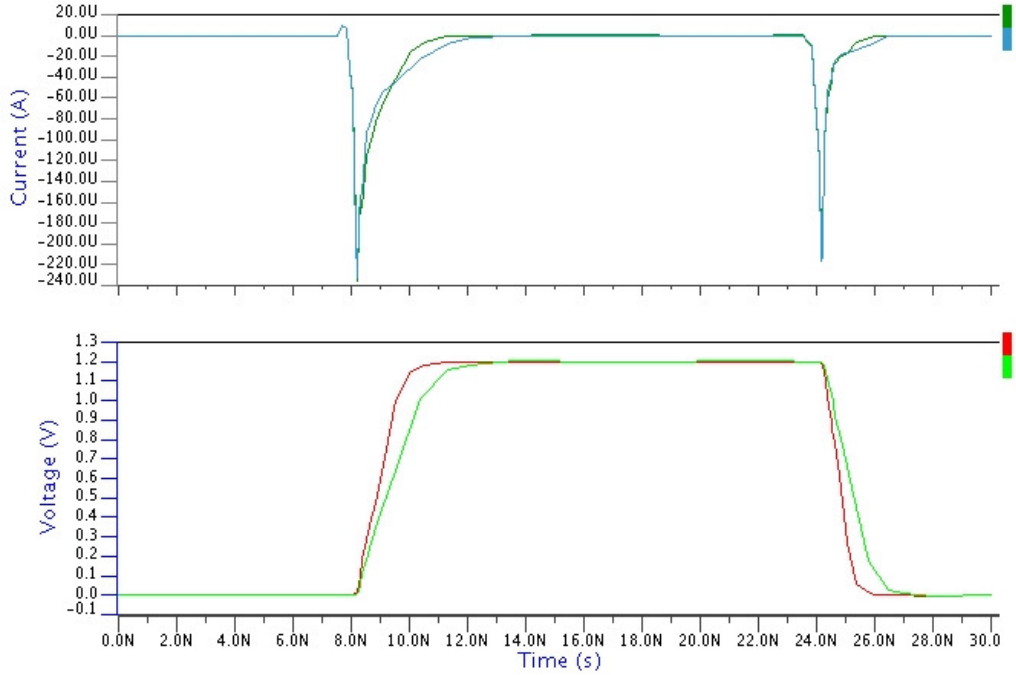


(a) Step current Response for 2×12 paths

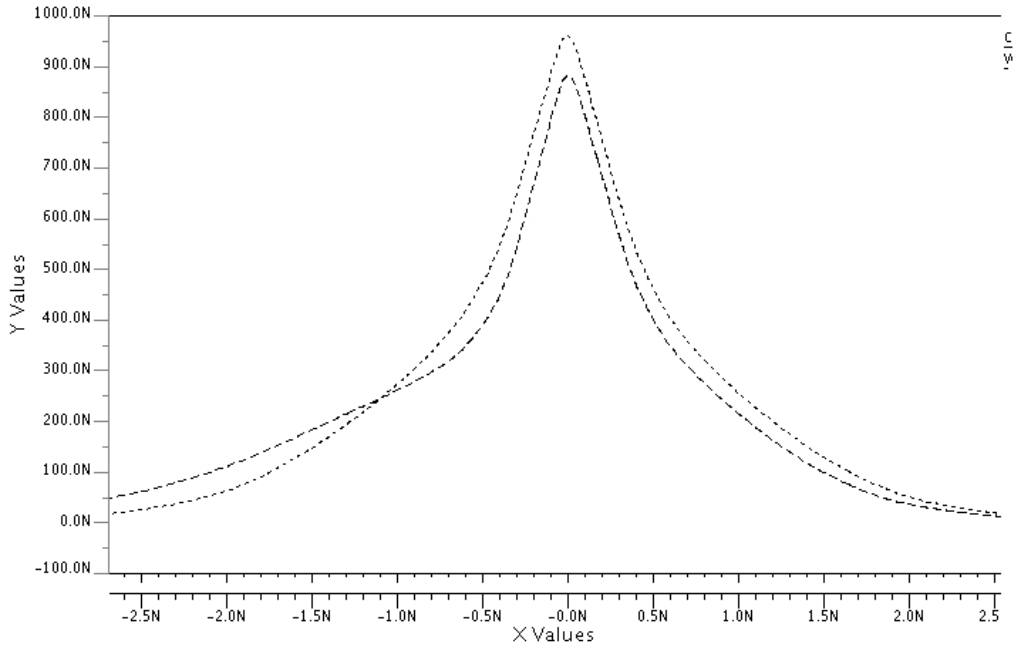


(b) Pair wise X-Correlation among C_2^{12}

Figure 7.20: Measuring Balancedness of each segment

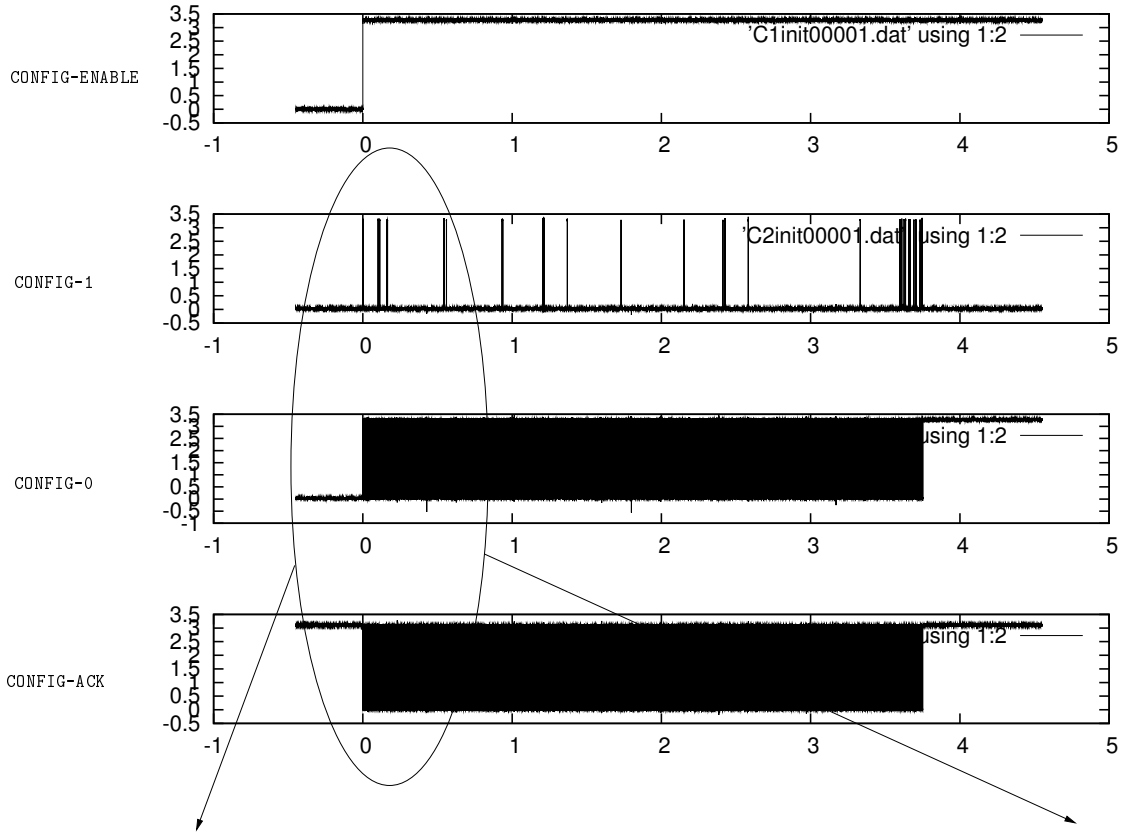


(a) delay and current difference for 1-hop difference

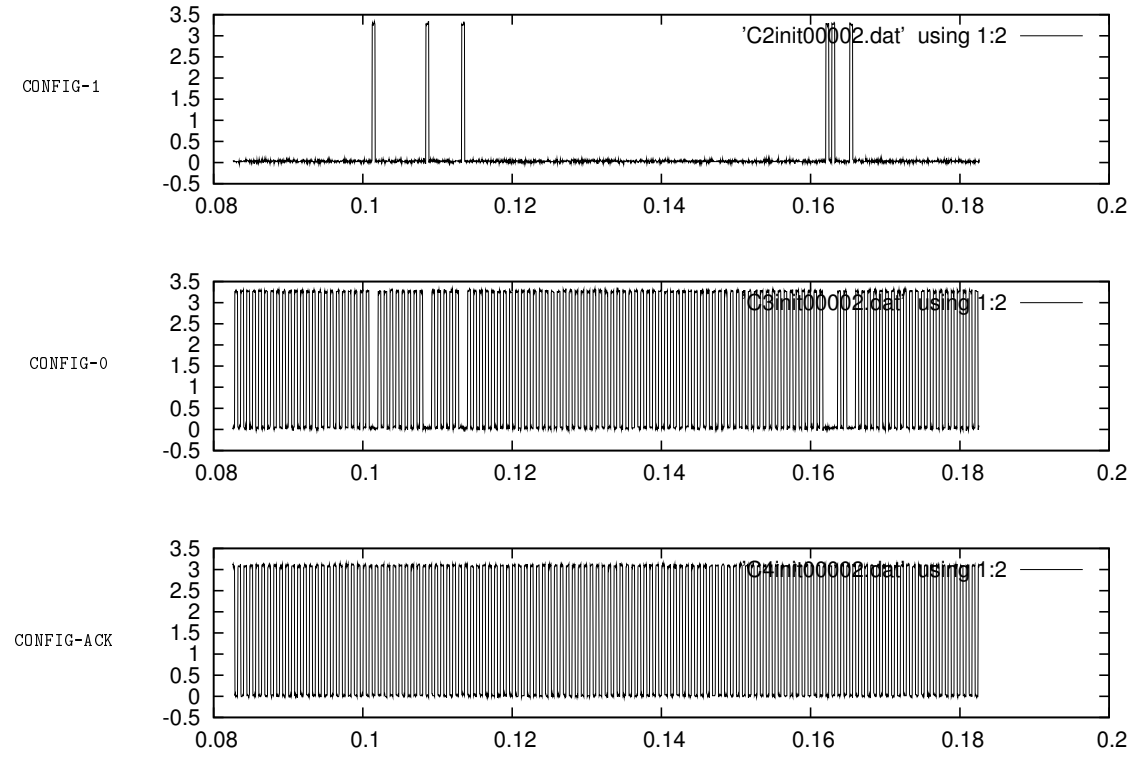


(b) X-Correlation for 0-hop mismatch and 1-hop mismatch

Figure 7.21: Measuring Hop Mismatch



(a) The Whole Configuration Trace 4692 bits



(b) Zoomed

Figure 7.22: Measured Trace of Configuration signals

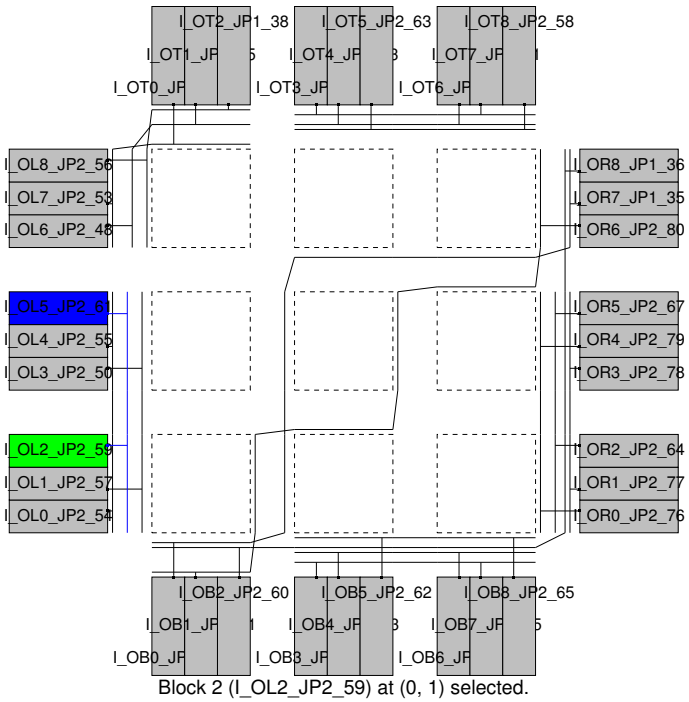


Figure 7.23: Routing form 6 inputs to 6 outputs configured in SAFE

7.9.2 Experiment: Interconnect

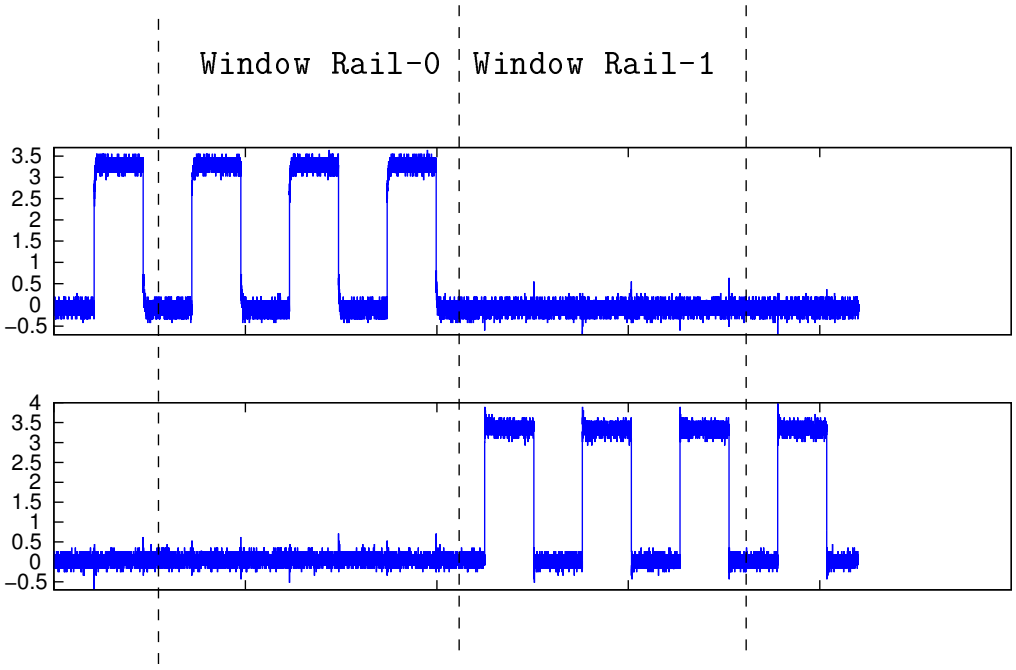
This experiment is only to test the functionality of the interconnects in the FPGA. For this purpose, we routed 6 inputs of the FPGA to 6 outputs of the FPGA SAFE (see fig. 7.23). All PLBs are configured with '0's. We used a very slow configuration speed, and sent random data on the input to read them back from the output, to validate that the configuration of the FPGA is correct and interconnects are functional.

The results of this experiment is positive, hence SAFE can be used later for other prototyping experiments, however the configuration speed is very slow and occasionally fails. With this experiment we can also measure the interconnect speed, and the balancedness achieved in silicon. This work is going on.

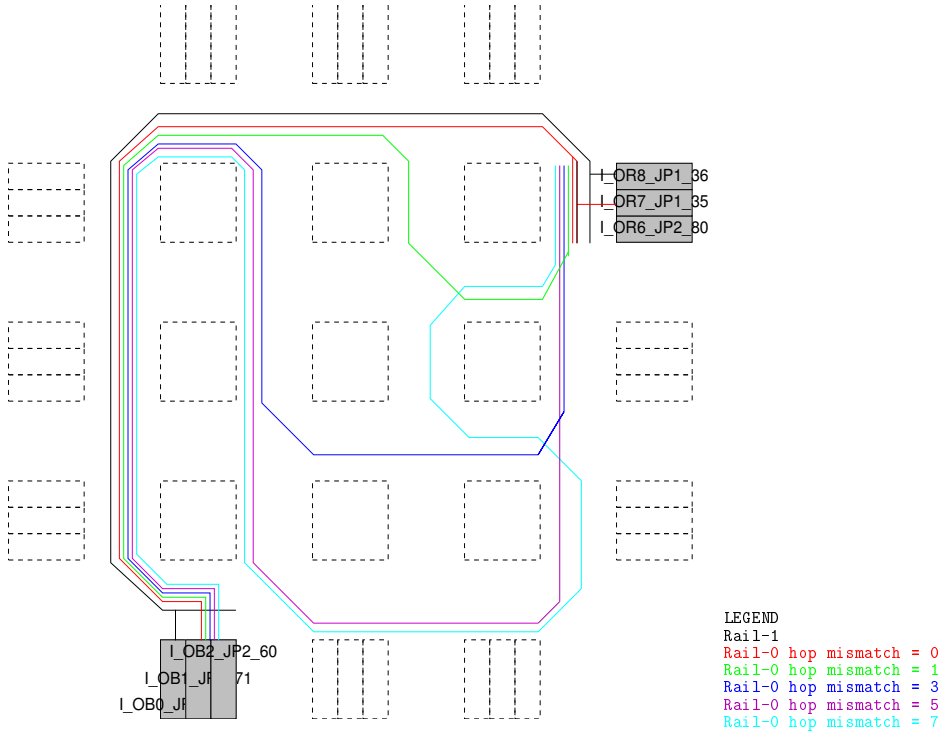
7.9.3 Experiment: Measuring Hop Mismatch

The goal of this experiment is to find out, the balancedness between the two rails constituting teh dual rail of asynchronous logic. This is very important in terms of robustness against side-channel attack since any difference between these two rails will give out the data values to the attacker. The Power-Constant methodology depends on this balancedness.

Figures 7.24(a) and 7.24(b) describe the experimental setup. We route a dual rail in the FPGA. The route for RAIL-1 of this dual rail stays constant over the experiment. The route for RAIL-0 is varied incorporating (0,1,3,5,7) difference in hops w.r.t RAIL-1, as shown in figure 7.24(b). We send 4 pulses each for RAIL-1 and RAIL-0, and we measure the power consumption. This measurement is done with a separate trigger signal encompassing 4+4 pulses. Among the 4 pulses we choose a window for the comparison. Let's say $W_1(t)$ is the trace for RAIL-1 and $W_0(t)$ is the trace for RAIL-0 within this



(a) Measuring the Difference between rail-0 and rail-1 for different lengths of rail-0



(b) After Processing

Figure 7.24: Description of the Experiment Hop Mismatch

Table 7.1: Hop-Mismatch in terms of RMS value

Hop Mismatch	$Balance_{RMS}$
0	1.106
1	1.120
3	1.158
5	1.119
7	1.173

Table 7.2: Hop-Mismatch in terms of X-Correlation

Hop Mismatch	$Balance_{XCORR}$
0	-0.613
1	-0.599
3	-0.661
5	-0.594
7	-0.671

window.

The balancedness between these two traces in terms of power consumption magnitude is then calculated as

$$Balance_{RMS} = \frac{RMS(W_0(t))}{RMS(W_1(t))}$$

Table 7.9.3 shows the results obtained from this experiment. We can see a visible increase in power consumption(RMS) with increasing Hop-Mismatch. The only exception being the value for hop mismatch 5. I guess this is an error in the measurement. So a repetition of this experiment needs to be done to verify this trend, however I omit it due to lack of time.

We also try to figure out the identicalness between these two traces in terms of power consumption patterns. So we use the following formula where a normalized cross correlation is taken between the two rails.

$$Balance_{Correlation} = \frac{Xcorr(W_1(t), W_0(t))}{Xcorr(W_1(t), W_1(t))}$$

The table 7.9.3 shows the experimental results. However from the data obtained we can not conclude since we don't see any specific trend. So as the hop-mismatch is increased the power consumption pattern is always different for the two rails but we don't see any proportional relationship between them.

The corresponding traces are shown in 7.25. We can see the triggering instants of the measurement. The traces for various hop mismatch and the balancedness values are indicated in 7.25. Figure 7.26 shows the same traces superposed in a staggered fashion for comparison purpose. They use the same color code as in fig. 7.24(b).

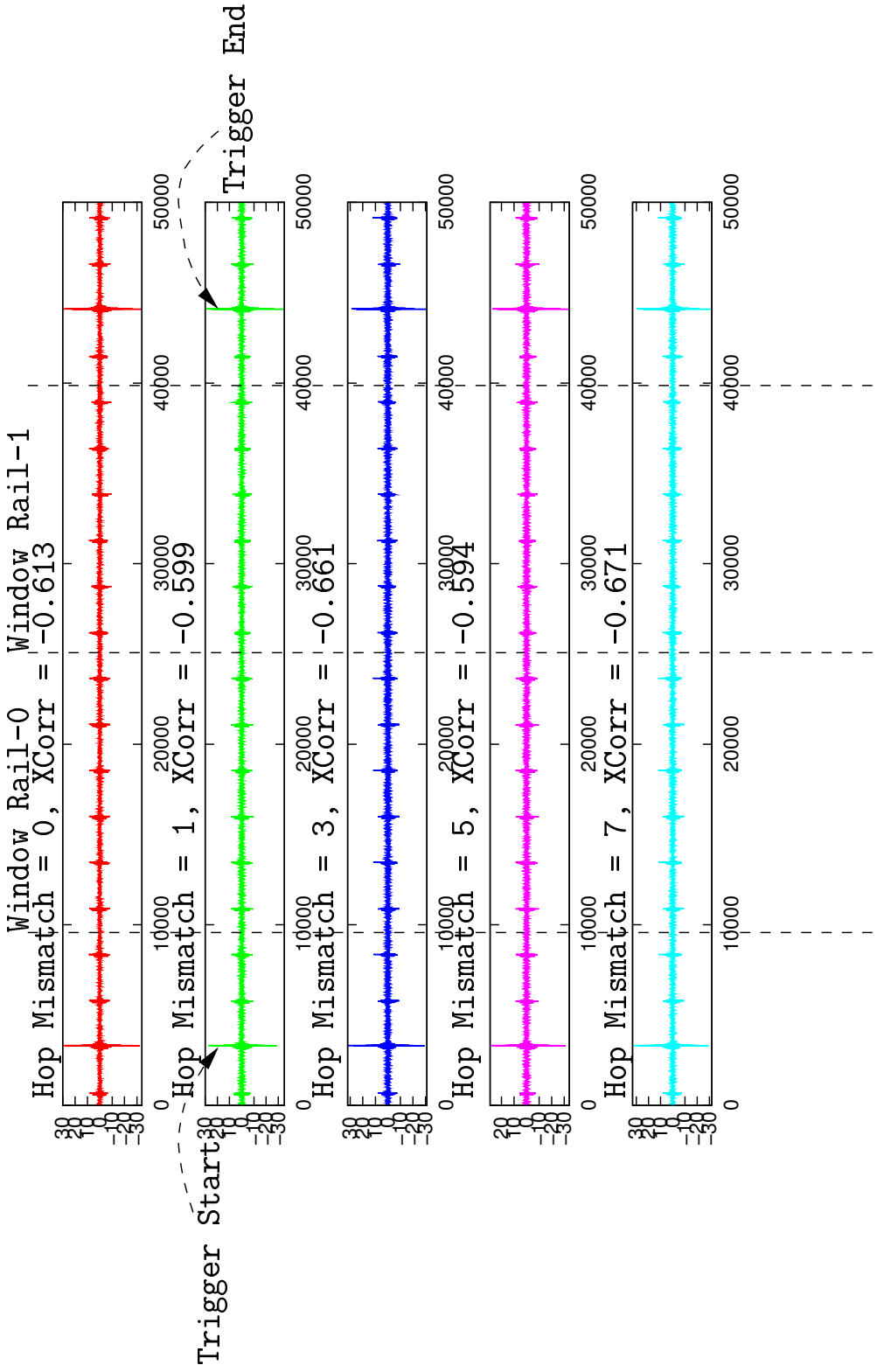


Figure 7.25: The results for different values of Hop Mismatch

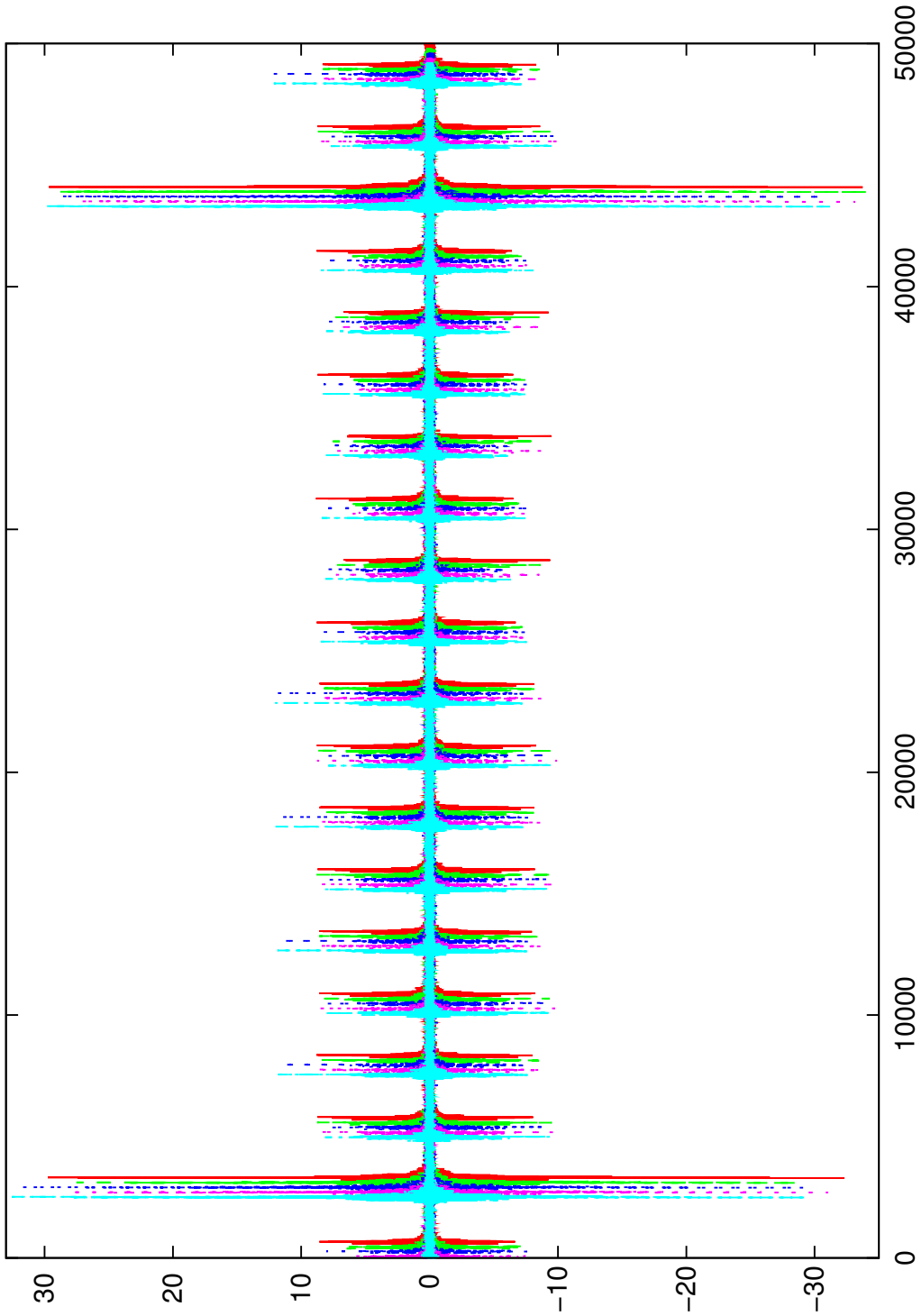


Figure 7.26: The results for different values of Hop Mismatch(Superposed and staggered)

7.9.4 Possible BUG

Due to the problems encountered during the configuration as explained in subsection 7.9.1, we investigated into the cause of this anomaly. We did the simulation of a single 6-input LUT inside the PLB. The implementation of this LUT is explained in subsection 7.3.0.4. In figure 7.27 we provide a more detailed view of the LUT configuration chain and switches. In actual silicon the memory points are connected to the LUT output through Transmission Gates (denoted as pass transistors in fig 7.27). The LUT inputs are decoded in a such a way that only one among these parallely connected Transmission Gates is "ON", depending on the input value, and the corresponding memory point should appear at the output.

However when the inputs go through a transition, there is a temporary short-circuit between the two concerned Transmission Gates(TGs). Because of the bidirectional nature of TGs, this disturbs the configuration chain itself. One LUT memory point is written into another one. This is illustrated in the simulation results in figure 7.28. In this experiment, we send a LUT bitstream of alternating '0's and '1's. The count signal is shown in figure 7.28. When the count signal reaches 64, the configuration is done. During this time we see alternating transitions on `config-in-0` and `config-in-1` and the corresponding `Config-ack-out` similar to those in figure 3.1(c). During this configuration phase the LUT inputs don't undergo any transition. As soon as the inputs make transitions, the corresponding memory points becomes '1' and the output is stuck at '1'. We also see that with input in transition the `config-out-0` and `config-out-1` assumes the invalid state ($'1',' 1'$).

We did the same simulation with tri-state buffers instead of Transmission Gates, in this case the output changes according to the inputs and memory points. We think that this could be a possible cause of the anomaly during the configuration of the FPGA "SAFE". In actual silicon, during the configuration, the LUT inputs are not forced to '0' or '1'. If the inputs go through parasitic transitions during configuration, this will introduce invalid state ($'1',' 1'$) in the configuration chain leading to random behaviour.

Further investigations, and testing is going on at TIMA, Grenoble, so that the FPGA can be used for basic testing, and this simulation results will be taken into account during the next tape out.

7.10 Secure Dual Rail Routing

As we have seen the previous chapter, the effect of hop-mismatch between dual-rails, we should guarantee balanced dual rail routing in CAD tools. We devised a simple dual rail routing algorithm where the routing tracks in FPGAs are divided into two domains, and each rail is then routed in separate domains. Because of the symmetrical domains, the routes for both rails are the same. However this will work only for homogeneous architectures, such as tracks with unit lengths, and subset switchbox. In table 7.3 we show the results of dual rail routings for some netlists in the QUIP [124]Benchmarks suite in a simple uniform architecture. The benchmarks are WDDL implementation of the netlists. As shown in table 7.3 we can guarantee a zero hop-mismatch routing with this technique. However this is only valid for simple homogeneous architectures, and needs considerable modification to be used in commercial architectures.

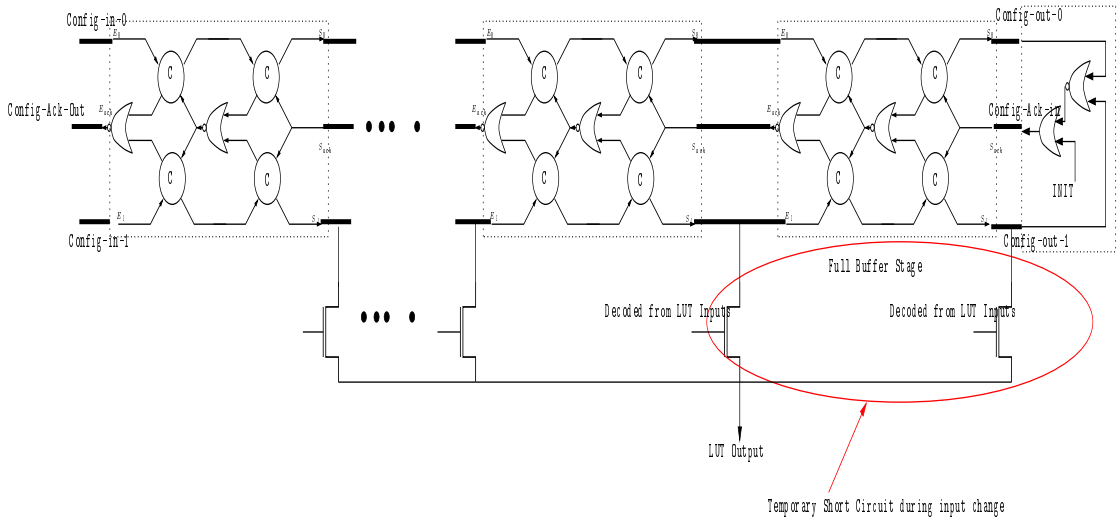


Figure 7.27: LUT implementation in detail

Table 7.3: Channel Width and Hop-Mismatch for Dual Rail Routing

Netlist	No. Nets	Breadth First		Dual Breadth First	
		Channel Width	Hop Mis-match /Dual-rail	Channel Width	Hop Mis-match /Dual-rail
barrel16_wddl	626	15	2.89	16	0
barrel32_wddl	1482	20	2.78	20	0
barrel64_wddl	3254	23	4.41	22	0
mux32_16bit_wddl	2964	12	0.83	12	0
mux64_16bit_wddl	5854	14	0.51	14	0
mux8_128bit_wddl	5932	11	2.66	12	0
mux8_64bit_wddl	2988	10	2.25	10	0
xbar_16x16_wddl	706	14	1.59	14	0

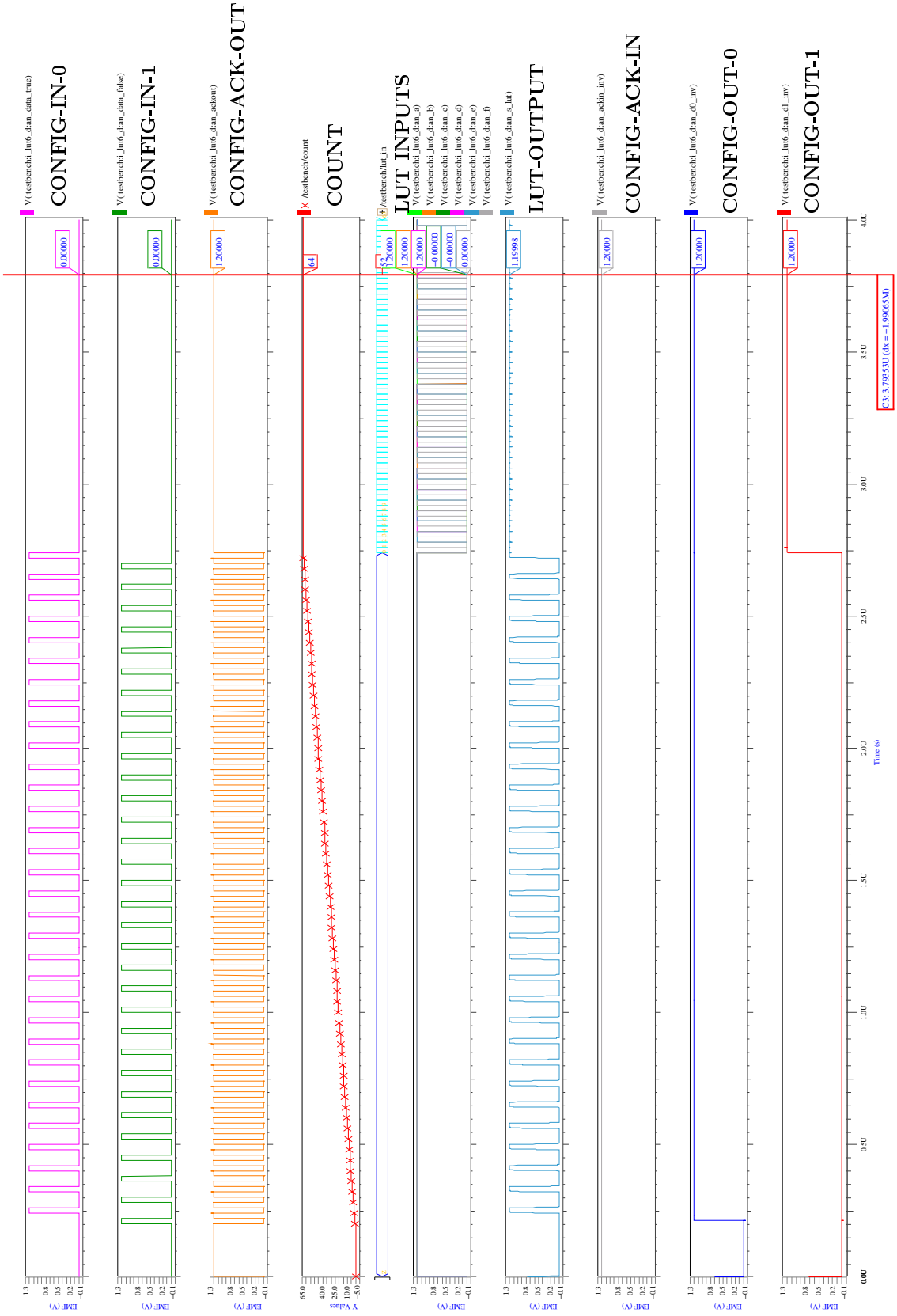


Figure 7.28: Simulation with Transmission Gates (Actual)

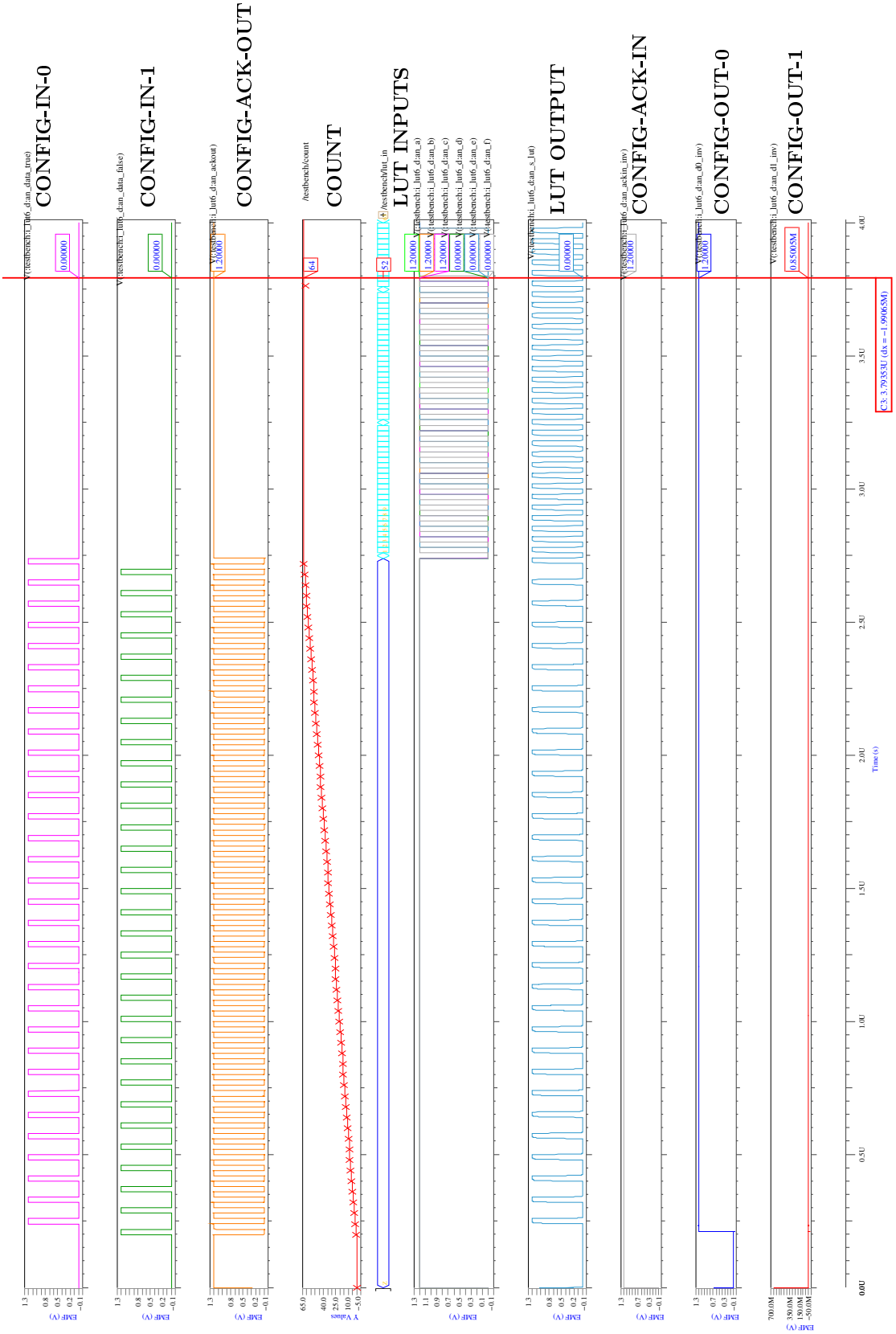


Figure 7.29: Simulation with tri-state buffers

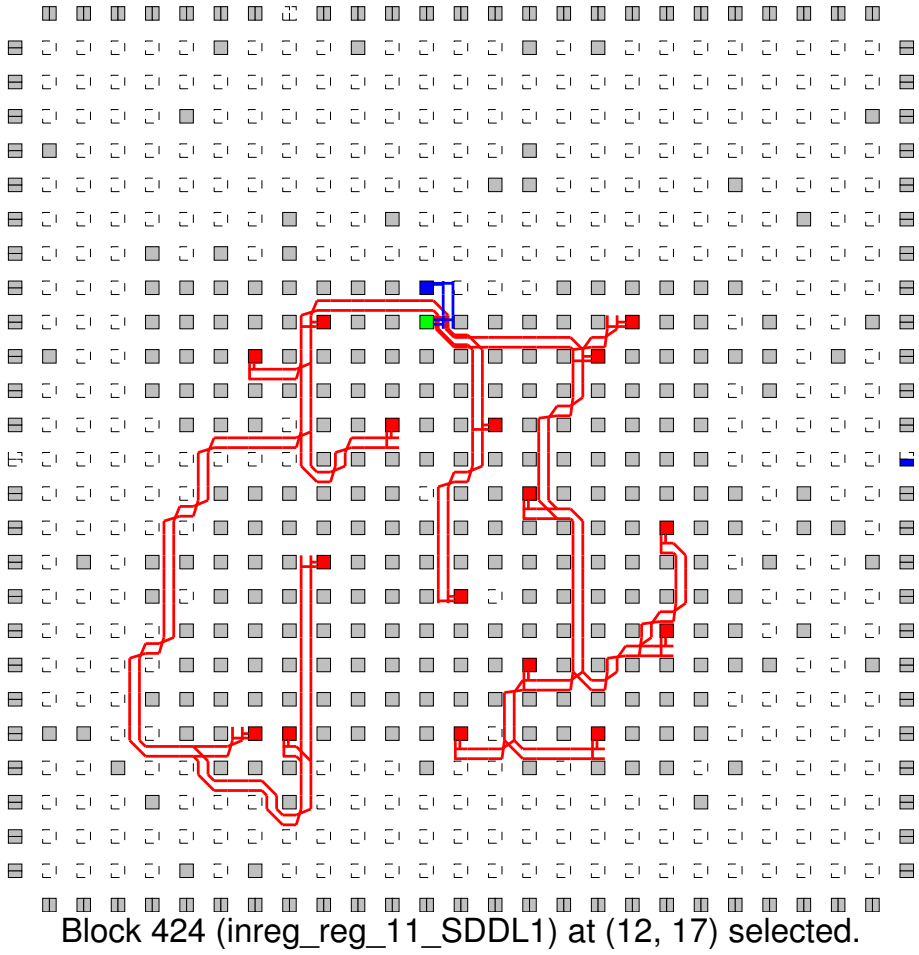


Figure 7.30: Dual Rail routing Example

7.11 Conclusion

In this chapter we discussed the suitability of an asynchronous FPGA as a counter-measure to the physical cryptanalysis, and as a prototyping device for such counter measures. Resistance of asynchronous circuits to fault injection, and power constant signal ling makes it a good candidate for such countermeasures. Moreover because of it's reconfigurable nature, it's possible to incorporate dynamic counter-measures along with static countermeasures. We believe a practical solution must use both of them to achieve highest level of security. We presented approximate models of power consumption and delay on which our counter measures are based, and defined our objectives for static countermeasures.

Keeping the prototyping role in mind, we presented a multi-style asynchronous PLB, and proposed a fine grain routing architecture, so that any m-out-of-n coding and various asynchronous protocols can be mapped onto this architecture. As shown in various experiments throughout the manuscript the power constant logical level protocols can't succeed without balanced interconnects. Layout statistics, and experiments on the extracted netlist from a prototype FPGA, presents the kind of balancedness in dynamic power consumption that can be achieved with the subset switchbox and the associated binary tree connection box. We also present a new physical implementation of the FPGA switch box called Tpair

switchbox, which provides indiscernabilty in EM emission for the dual rails routed through it.

Although the solutions proposed in actual layout assume bidirectional FPGA interconnect, we show how these solutions can be ported to other flavours of interconnect such as single-driver, both at the switchbox and connection box level. Finally, the 3x3 prototype asynchronous FPGA in 65 nm(CMP Run S_C5C81) will help us to evaluate the mix of dynamic countermeasures at application design time, with the static countermeasures that we incorporated into it during circuit design time. For example swapping of the constituting rails of a dual-rail signal between each run. Last but not the least, we would also like to stress the importance of CAD algorithms in physically securing the application (e.g automatically routing dual rails through this FPGA in a balanced way). These important issues are the main future research challenges.

Part III

Conclusion & Perspective

Summary

The last part of this thesis manuscript presents the general conclusions and in chapter 8 we propose limiting of cryptographic device lifetime using Auto-Destructive Circuits as a future research direction.

Conclusion

This thesis manuscript touches upon three very different domains in the modern computer architecture field, namely *Cryptography*, *FPGA architectures* and *Asynchronous Circuits*, with the specific goal of finding a suitable counter-measure to physical cryptanalysis attacks on cryptographic hardware.

In the first part of this manuscript, we have seen a brief description of existing attacks and related countermeasures classified according to their MTD (Measurements to Disclosure), which we broadly divided into static or dynamic countermeasures. Static countermeasures are incorporated during the circuit design time, and often relies on power constant signalling, that is power consumption is independent of underlying computation. These countermeasures requires a significant effort in circuit design(i.e guaranteeing minimum mismatch both in transistors and interconnect). On the other hand, Dynamic counter-measures try to randomize computation instants, or masking the profile of one computation by a complementary one. They have some reconfiguration overhead, due to particular sequence/protocol used, but can evolve even after circuit fabrication.

This thesis is an effort to reconcile these two approaches, by combining there advantages, and efficiently managing the combined disadvantages. To achieve this goal, we fabricated two prototype FPGAs.

- An embedded FPGA in 130nm technology which have been tested and demonstrates partial and Run-Time Reconfiguration(*RTR*), with correct functionality. Because of its reconfigurable nature this eFPGA can be used in future to evaluate the feasibility of countermeasures using *RTR*.

- An Fully asynchronous FPGA in 65nm technology in collaboration with *TIMA, Grenoble* suitable for evaluating both dynamic and asynchronous countermeasures, and multiple asynchronous protocols. This stand-alone FPGA has been tested and shows correct functionality of the asynchronous configuration chain and reset mechanism, as well as correct functionality for interconnects. However we have identified a possible bug in the implementation(see section 7.9.4 for details), because of which couldn't test the asynchronous PLBs, and it also limits the speed of asynchronous configuration chain. Further tests are going on at TIMA, Grenoble. Upon confirmation, this bug will be corrected in the next run, and for the time being we present results of transistor and interconnect mismatch based on spice simulation with extracted parasitics(STARRCXT [121]). The simulation results are well in accordance with test results, at least for interconnects.

The present conclusion and future research directions are based on the experimental results as well as design flow complexity described in section two and three. In brief my conclusion is that a mix of dynamic and static countermeasures is necessary to achieve

the maximum MTD(Measurements to disclosure: a popular metric used to rank counter-measures) and finally some decay mechanism(e.g EM, NBTI) must be used to limit the number of times a cryptographic device can be used, evidently less than it's MTD. From this point of view, the most important question is an accurate determination of MTD, the counter-measures are only used to increase the lifetime of a cryptographic device.

To elaborate more on the above statement: a successful implementation of static (power constant signaling) counter-measure requires balanced interconnects. While this may be possible in principle, achieving identical length, shape, and neighbourhood(for parasitics) in presence of sub-lithographic design rules is very complicated with only marginal improvements. So the author's point of view is to achieve a reasonable amount of indiscernability in silicon, without going into very complicated design/verification flows. and later cancel the residual difference with dynamic counter-measures. for example a simple dynamic counter-measure is to alternate the routes for signal 0 and signal 1 between each encryption. The auto-decay of a cryptographic device (based on EM,NBTI) is a future research direction. The main challenges are prediction of the lifetime of a circuit and controlling these decay mechanisms, so that it never crosses the MTD.

Chapter 8

Perspective: Auto-Destructive Circuits

As we have seen in the previous sections of this thesis manuscript, all countermeasures to Side-Channel Attacks are classified according to their MTD(Measurement to Disclosure). Ideally a counter measure should have a MTD so high that it is not possible for an attacker to get the secret key in reasonable amount of time and space.

Hence one possible research direction is to be continuously on the lookout for countermeasures with high MTD, which has been the guiding principle of this thesis. Here we mention continuously because the MTD of a countermeasure can change significantly, depending on the advancement of the acquisition equipments, and efficient post-processing techniques.

Another possible research direction is to limit the number of utilization of a cryptographic device beforehand. In all the known attacks, the attacker has to use the device a certain number of times guess the key, or build a template of the concerned device. The main research challenges in this approach is to accurately predict and ability to tune the device lifetime, and hence an increased comprehension and control of the decay processes. The reader may note that, we can do so by use of digital counters in the circuit, however in that case the attackers will try to hack these counters. To mitigate this potential attack the author proposes bounding the life time of cryptographic devices based on intrinsic decay mechanisms [131]. We briefly describe in this section, some of the possible physical processes that we can exploit to achieve such bounded life-times.

8.1 Electromigration

Electromigration is the term applied to the transport of metal atoms due to the momentum exchange with the "Electron Wind". Due to the Joule heating of wires, some metal atoms could be thermally excited out of it's minimum energy configuration, and are free to move out of the metal lattice. The momentum exchange between the conducting electrons colliding with the activated metal atoms("Electron wind") displaces the metal atom out of the lattice and leaves a vacancy. The accumulation of vacancies could finally lead to an open circuit in the interconnecting wire.

This Electromigration process is governed by Black's Equation [127]

$$\frac{w \times t}{MTF} = AJ^2 \exp \frac{\phi}{kT} \tag{8.1}$$

Where

- w is the conductor width in cms.
- t is the conductor thickness in cms.
- A is a constant depending on several physical properties of the material such as volume resistivity, average electron free path etc.
- ϕ is the activation energy in Electron Volts.
- k is Boltzmann's Constant $8.62 \times 10^{-5} ev/K$.
- T is temperature in degree Kelvin.
- MTF is Mean Time to Failure.

8.2 Negative Bias Temperature Instability(*NBTI*)

Negative Bias Temperature Instability is the process of generation of interface traps under negative bias($V_{gs} = -V_{dd}$) at elevated temperature in PMOS transistors This process of is an important issue for sub-100nm technologies due to increasing electric field across the thin $Si - SiO_2$ boundary. NBTI results in an increase of threshold voltage(V_{th}) with time, leading to failure.

The NBTI decay follows the experimentally determined relationship with time(t) [131]

$$\Delta V_{th} = (K^2 t^{0.5} + c)^{0.5} \tag{8.2}$$

This decay process can be further enhanced by scaling gate-oxide thickness, or using different gate oxide materials. Reference [131] proposes an auto-destructive circuit using *NBTI*. Recently published refernce [129] also proposes a method of metering based on hardware aging using *NBTI*.

8.3 Memristor

Memristor is a new device which has created a lot of ripples in the semiconductor research in recent times. First conceptualized by Leon Chua [128], first prototypes of this device has recently been fabricated. Memristor seems to be a device with very promising applications, such as implementation of neural networks, and mass memory.

The characteristic of a memristor is that it resistance increases/decreases as a function of current that has passed through the resistor. Because of this reason, this could be used to memorize the number of times a circuit has been used. For example, by placing a carefully designed memristor in the power supply line, we can increase the supply resistance incrementally, with each use of the device, and finally leading to an open circuit before its MTD. Memristor technology is at a very early stage of research, and it will require considerable time before it can be used reliably in a working circuit.

Part IV

Miscellaneous

Glossary

AES: Advanced Encryption System.
CAD: Computer Aided Design.
CHP: Communicating Hardware Processes.
CLB: Configurable Logic Block.
DES: Data Encryption System.
DPA: Differential Power Analysis.
EMA: Electro-Magnetic Analysis.
FASE: FASE is not "SAFE".
FPGA: Field Programmable Gate Arrays.
HOS: Hardware Operating System.
IP: Intellectual Property Block.
LCM: Least Common Multiple.
LUT: Look-Up Table.
MDPL: Masked Dual-Rail with Precharged Logic.
MTD: Measurements to Disclosure.
PLB: Programmable Logic Block.
RTR: Run-Time Reconfigurable.
SAFE: Secure Asynchronous FPGA for Embedded systems.
SCA: Side-Channel Attacks.
VCI: Virtual Component Interface.
VPR: Versatile Place & Route.

Notations

8.4 Useful Results

In this section we put together all essential formulas and notations(see Table 2.1) used in our calculations. Equations 8.3 and 8.4 are from Donath's manuscript [62] and others are derived with Maxima [132].

$$\sum_{i_A=1}^{W}\sum_{j_A=1}^{W}\sum_{i_B=1}^{W}\sum_{j_B=1}^{W} W + i_B - i_A + |j_B - j_A| = \frac{4}{3}W^5 - \frac{1}{3}W^3 \tag{8.3}$$

$$\sum_{i_A=1}^{W}\sum_{j_A=1}^{W}\sum_{i_B=1}^{W}\sum_{j_B=1}^{W} 2W + i_A + j_A - i_B - j_B = 2W^5 \tag{8.4}$$

$$\sum_{x=-W}^{W}\sum_{y=0}^{W} (W-|x|)(W-|y|)\,|W+x-|y|| = \frac{21W^5+30W^4+5W^3+4W}{60} \tag{8.5}$$

$$\sum_{x=-W}^{W}\sum_{y=-W}^{-1} (W-|x|)(W-|y|)\,|W+x-|y|| = \frac{21W^5-30W^4+5W^3+4W}{60} \tag{8.6}$$

$$\sum_{x=-W}^{W}\sum_{y=-W}^{W} (W-|x|)(W-|y|)\,|W+x-|y|| = \frac{42W^5+10W^3+8W}{60} \tag{8.7}$$

$$\sum_{x=-W}^{W}\sum_{y=0}^{W} (W-|x|)(W-|y|)\,[W+x+|y|] = \frac{4W^5+3W^4-W^3}{6} \tag{8.8}$$

$$\sum_{x=-W}^{W}\sum_{y=-W}^{-1} (W-|x|)(W-|y|)\,[W+x+|y|] = \frac{4W^5-3W^4-W^3}{6} \tag{8.9}$$

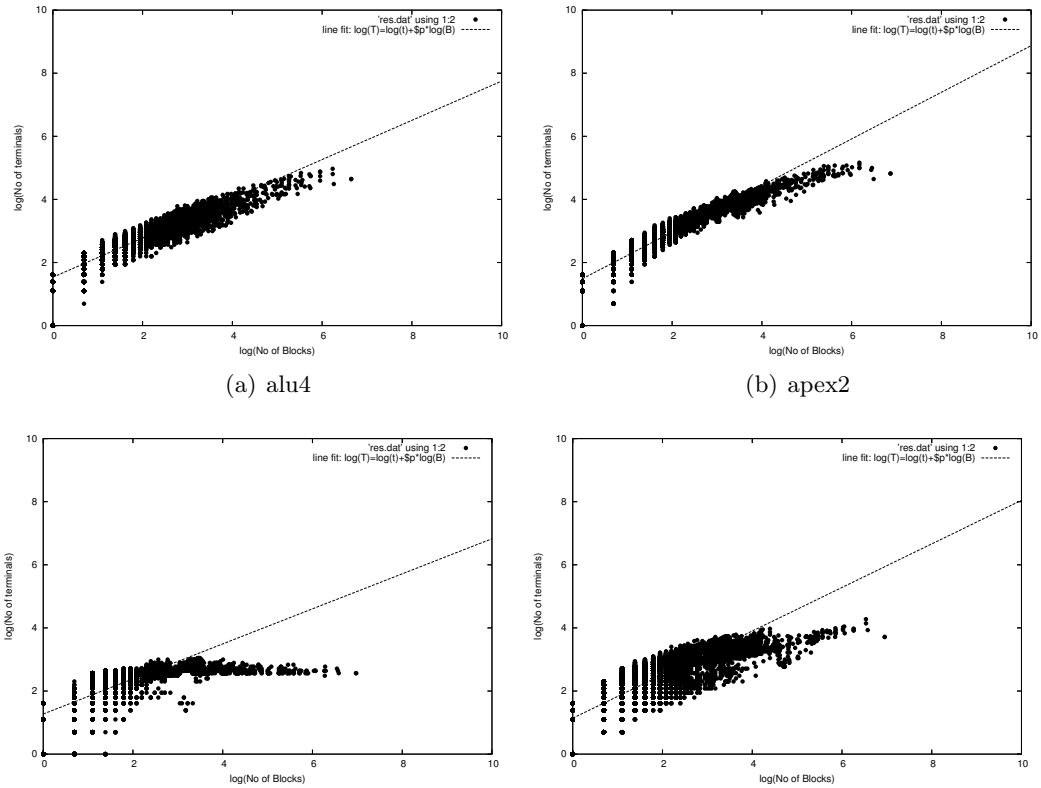
$$\sum_{x=-W}^{W}\sum_{y=-W}^{W} (W-|x|)(W-|y|)\,|x-y| = \frac{7W^5-5W^3-2W}{15} \tag{8.10}$$

$$\sum_{i_A=1}^{W}\sum_{i_B=1}^{W} f(i_A - i_B) = \sum_{i_{AB}=-W}^{W} (W-|i_{AB}|)\times f(i_{AB}) \tag{8.11}$$

$$\sum_{k=0}^{L-1}(a+kd)r^k = \frac{a(1-r^L)}{1-r} + \frac{rd\left[1-Lr^{L-1}+(L-1)r^L\right]}{(1-r)^2} \tag{8.12}$$

Appendix A
Rent Plot for MCNC Benchmarks

8.5



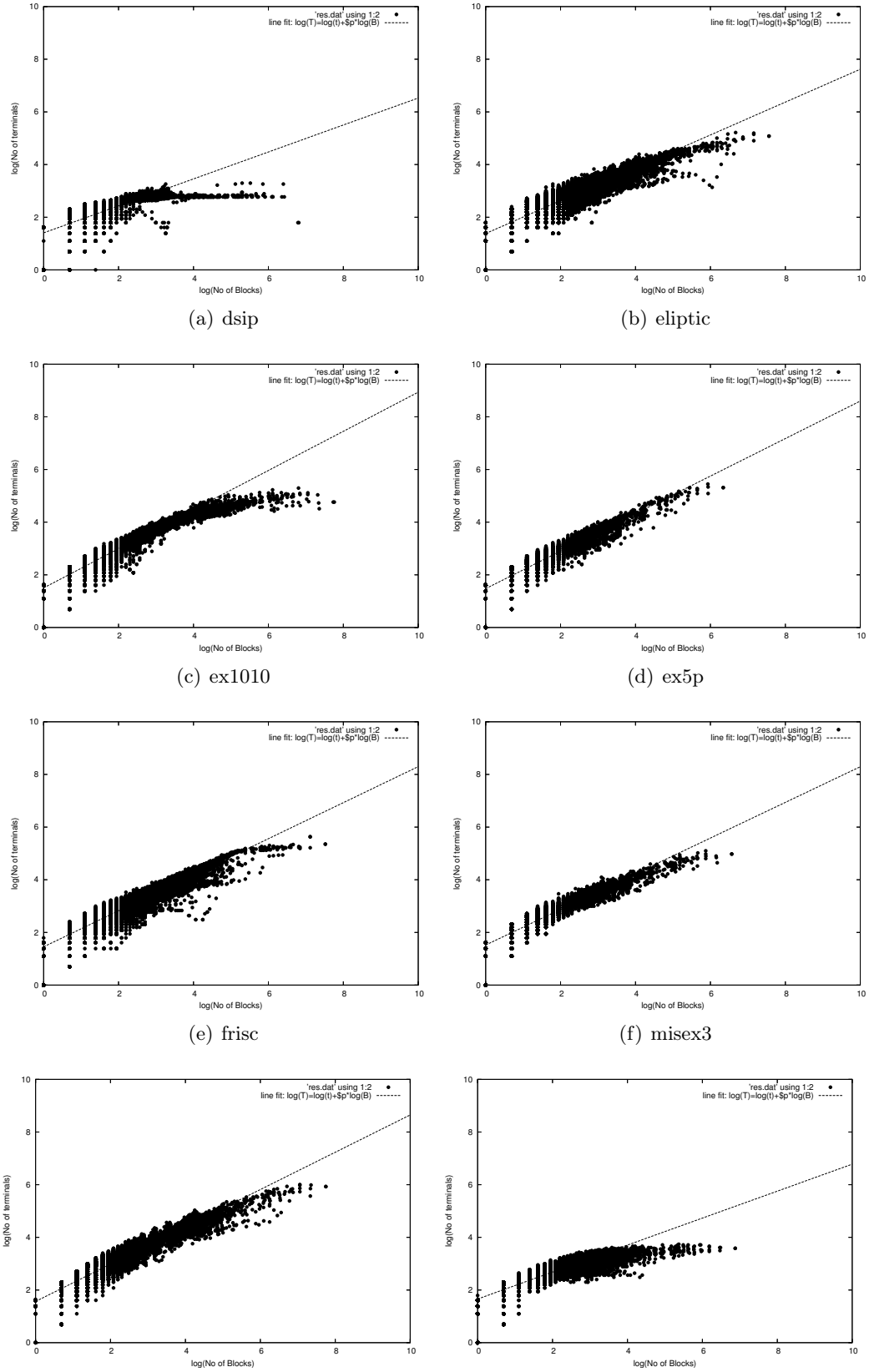
(a) alu4

(b) apex2

(c) bigkey

(d) des

Figure 8.1: Rent Plots for MCNC Benchmarks



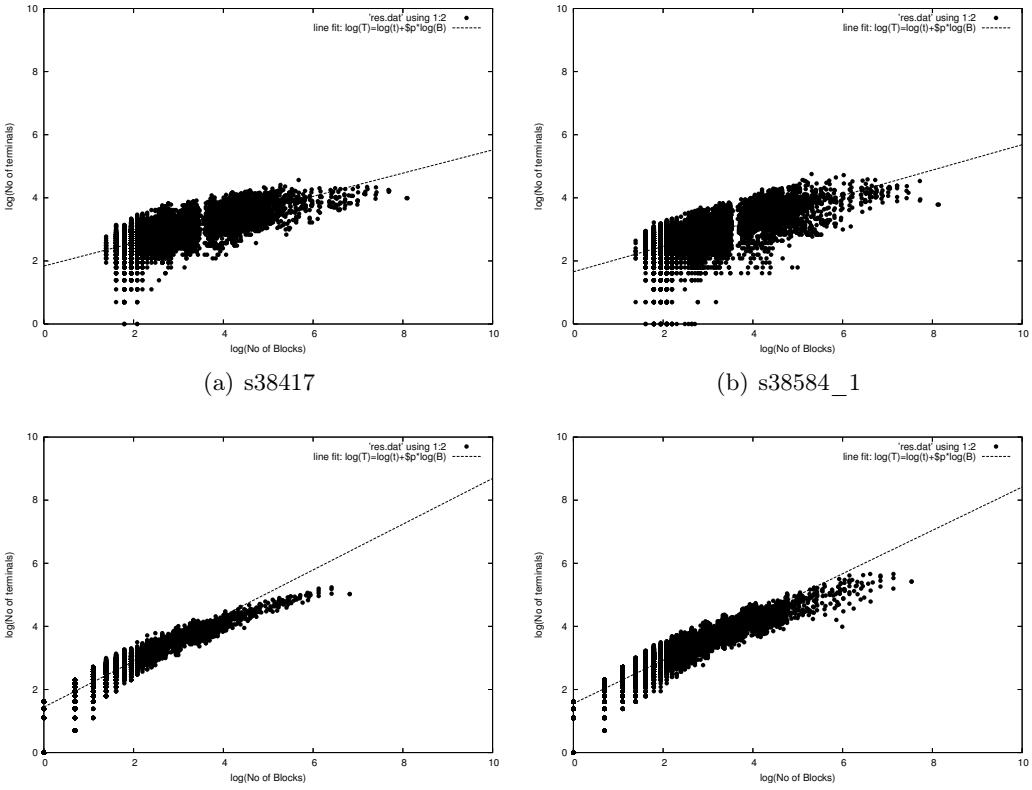
(a) dsip (b) eliptic

(c) ex1010 (d) ex5p

(e) frisc (f) misex3

(g) pdc (h) s298

Figure 8.2: Rent Plots for MCNC Benchmarks



(a) s38417

(b) s38584_1

(c) seq

(d) spla

Figure 8.3: Rent Plots for MCNC Benchmarks

Appendix B
Test Setup: FASE

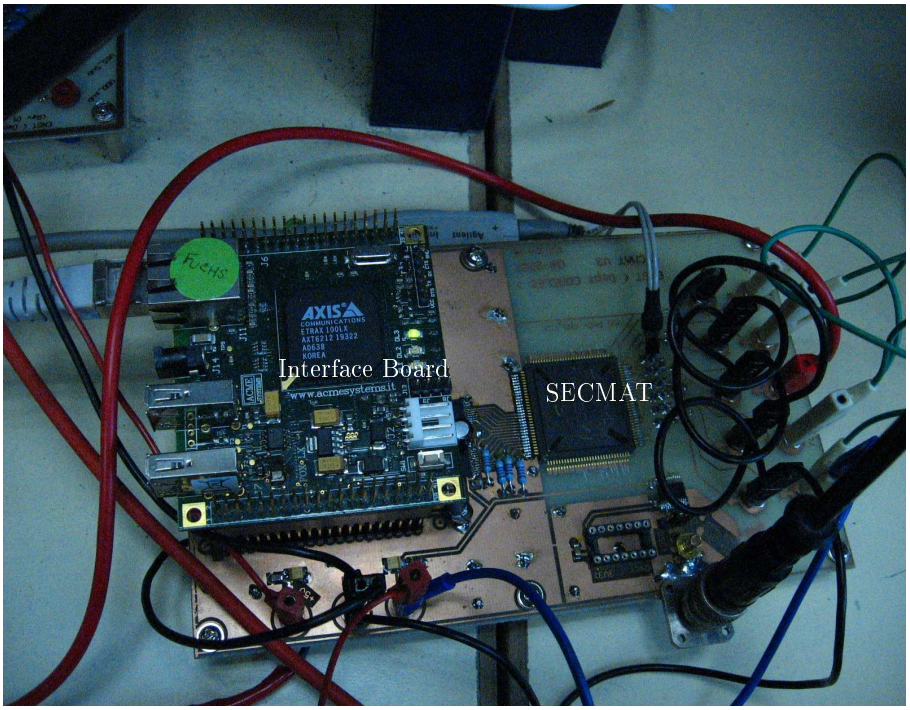


Figure 8.4: FASE (embedded into the ASIC "SECMAT") Test Setup.

Appendix C
Test Setup: SAFE

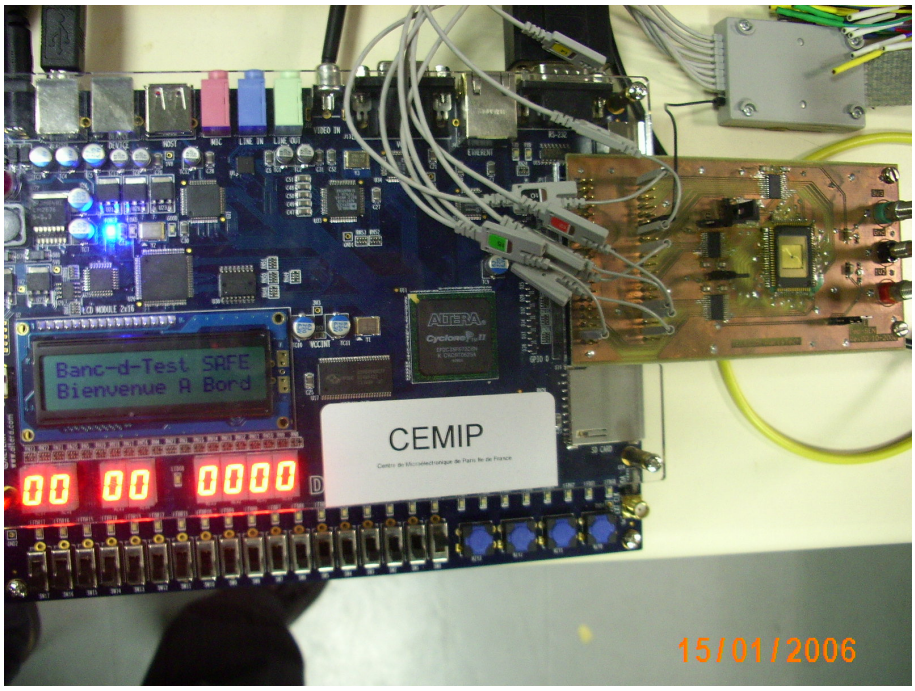


Figure 8.5: SAFE Test Setup.

Bibliography

Bibliography: Crypto

[1] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The EM Side-Channel(s). In *CHES*, volume 2523 of *LNCS*, pages 29–45. Springer, 2002. (document)

[2] Mehdi-Laurent Akkar and Christophe Giraud. An Implementation of DES and AES Secure against Some Attacks. In LNCS, editor, *Proceedings of CHES'01*, volume 2162 of *LNCS*, pages 309–318. Springer, May 2001. Paris, France. 1.3.3.1, 4.2

[3] Ross J. Anderson and Markus G. Kuhn. Tamper Resistance – a Cautionary Note. In *The Second USENIX Workshop on Electronic Commerce*, November 18–21 1996. Oakland, California; ISBN 1-880446-83-9. (Online HTML version). 6.4.1

[4] Cédric Archambeau, Eric Peeters, François-Xavier Standaert, and Jean-Jacques Quisquater. Template attacks in principal subspaces. In *CHES*, pages 1–14, 2006. 1.3.1.2, 1.3.2.4

[5] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, 94(2):370–382, Feb. 2006. (document)

[6] Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. The Sorcerer's Apprentice Guide to Fault Attacks. Cryptology ePrint Archive, report 2004/100. 6.4.3

[7] Eli Biham and Adi Shamir. Differential Cryptanalysis of the Full 16-Round DES. In *CRYPTO '92: Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, pages 487–496, London, UK, 1993. Springer-Verlag. (document), 1.2.1.1

[8] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *CHES*, pages 16–29, 2004. 1.3.2.3

[9] Don Coppersmith. The Data Encryption Standard (DES) and its strength against attacks. *IBM Journal of Research and Development*, 38(3):243–250, 1994. 1.2.1.1

[10] Wieland Fischer and Berndt M. Gammel. Masking at Gate Level in the Presence of Glitches. In *Cryptographic Hardware and Embedded Systems -CHES 2005*, pages 187–200, 2005. 4.4

[11] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic Analysis: Concrete Results. In *CHES*, volume 2162 of *LNCS*, pages 251–261. Springer, May 2001. (document)

[12] Louis Goubin and Jacques Patarin. DES and Differential Power Analysis (The "Duplication" Method). In *Proc. of CHES*, volume LNCS 1717, pages 158–172. Springer, 1999. 6.4.2

[13] S. Guilley and J-L. Danger. Test de circuits cryptographiques par analyse différentielle de consommation. (FR 08 51184), mar 2008. 1.3.1.2

[14] S. Guilley, Ph. Hoogvorst, Y. Mathieu, R. Pacalet, and J. Provost. CMOS Structures Suitable for Secured Hardware. In *Proceedings of DATE'04*, pages 1414–1415. IEEE Computer Society, February 2004. Paris, France. 1.3.3.2

[15] Sylvain Guilley, Philippe Hoogvorst, Yves Mathieu, and Renaud Pacalet. The "Backend Duplication" Method. In *CHES*, volume LNCS 3659, pages 383–397. Springer, 2005. August 29th – September 1st, Edinburgh, Scotland, UK. 1.3.3.2

[16] M. Hellman. An overview of public key cryptography. *Communications Magazine, IEEE*, 16(6):24–32, Nov 1978. 1.1.2.2

[17] K. Tiri and I. Verbauwhede. Place and Route for Secure Standard Cell Design. In *Proceedings of WCC / CARDIS*, pages 143–158, Aug 2004. Toulouse, France. 1.3.3.2

[18] Mark Zwolinski Karthik Baddam. Path switching: a technique to tolerate dual rail routing imbalances. 12(3):207–220, 2008. http://www.springerlink.com/content/32181g28411w2121. 1.3.3.1

[19] Auguste Kerckhoffs. La cryptographie militaire (1). *Journal des sciences militaires*, 9:5–38, January 1883. http://en.wikipedia.org/wiki/Kerckhoffs_law. 8.5

[20] Auguste Kerckhoffs. La cryptographie militaire (2). *Journal des sciences militaires*, 9:161–191, February 1883. http://en.wikipedia.org/wiki/Kerckhoffs_law. 8.5

[21] P. Kocher, J. Jaffe, and B. Jun. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Proceedings of CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996. (document)

[22] P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis: Leaking Secrets. In *Proceedings of CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer-Verlag, 1999. (document), 6.4.2

[23] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis: Leaking Secrets. *Advances in Cryptology: Proceedings of CRYPTO'99*, LNCS 1666:388–397, august 1999. Cryptology Conference, Santa Barbara, California, USA. (PDF). 4.1

[24] François Mace, François-Xavier Standaert, Ilham Hassoune, Jean-Jacques Quisquater, and Jean-Didier Legat. A Dynamic Current Mode Logic to Counteract Power Analysis Attacks. In *DCIS 2004 - 19th Conference on Design of Circuits and Integrated Systems*, pages 186 – 191, 2004. 1.3.3.2

[25] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully Attacking Masked AES Hardware Implementations. In LNCS, editor, *Proceedings of CHES'05*, volume 3659 of *LNCS*, pages 157–171. Springer, September 2005. Edinburgh, Scotland, UK. 4.2, 4.4

[26] Mitsuru Matsui. Linear cryptanalysis method for des cipher. In *EUROCRYPT '93: Workshop on the theory and application of cryptographic techniques on Advances in cryptology*, pages 386–397, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc. (document), 1.2.1.1

[27] Daniel Mesquita, Jean-Denis Techer, Lionel Torres, Gilles Sassatelli, Gaston Cambon, Michel Robert, and Fernando Moraes. Current mask generation: a transistor level security against dpa attacks. In *SBCCI '05: Proceedings of the 18th annual symposium on Integrated circuits and system design*, pages 115–120, New York, NY, USA, 2005. ACM. 1.3.3.2

[28] Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Investigations of power analysis attacks on smartcards. In *WOST'99: Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology*, pages 17–17, Berkeley, CA, USA, 1999. USENIX Association. 1.3.2.3

[29] Robert Morris and Ken Thompson. Password security: a case history. *Commun. ACM*, 22(11):594–597, 1979. 1.1.2.1

[30] NIST/ITL/CSD. Data Encryption Standard. FIPS PUB 46-3, Oct 1999. http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf. 1.1.2.1, 4.1.1

[31] NIST/ITL/CSD. Advanced Encryption Standard (AES). FIPS PUB 197, Nov 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf. 1.1.2.1

[32] Siddika Berna Örs, Frank Gürkaynak, Elisabeth Oswald, and Bart Preneel. Power-Analysis Attack on an ASIC AES implementation. In *ITCC '04: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04) Volume 2*, page 546, Washington, DC, USA, 2004. IEEE Computer Society. (document)

[33] Elisabeth Oswald, Stefan Mangard, Norbert Pramstaller, and Vincent Rijmen. A Side-Channel Analysis Resistant Description of the AES S-box. In LNCS, editor, *Proceedings of FSE'05*, volume 3557 of *LNCS*, pages 413–423. Springer, February 2005. Paris, France. 1.3.3.1, 4.2

[34] Éric Peeters, François-Xavier Standaert, Nicolas Donckers, and Jean-Jacques Quisquater. Improved Higher-Order Side-Channel Attacks with FPGA Experiments. In *CHES*, volume 3659 of *LNCS*, pages 309–323. Springer, 2005. 1.3.3.1

[35] Thomas Popp and Stefan Mangard. Masked Dual-Rail Pre-charge Logic: DPA-Resistance Without Routing Constraints. In LNCS, editor, *Proceedings of CHES'05*, volume 3659 of *LNCS*, pages 172–186. Springer, Sept 2005. Edinburgh, Scotland, UK. 1.3.3.2

[36] Alin Razafindraibe, Michel Robert, and Philippe Maurine. Analysis and Improvement of Dual Rail Logic as a Countermeasure Against DPA. In *PATMOS*, pages 340–351, 2007. Göteborg, Sweden. 1.3.3.2

[37] Christian Rechberger and Elisabeth Oswald. Practical Template Attacks. In *WISA*, volume 3325 of *LNCS*, pages 443–457. Springer, august 2004. (document), 1.2.1.2

[38] SCARD European FP6 project website: http://www.scard-project.eu. 4.2

[39] Patrick Schaumont and Kris Tiri. Masking and dual-rail logic don't add up. In *CHES '07: Proceedings of the 9th international workshop on Cryptographic Hardware and Embedded Systems*, pages 95–106, Berlin, Heidelberg, 2007. Springer-Verlag. 1.3.3.1, 1.3.3.2

[40] Patrick Schaumont and Kris Tiri. Masking and Dual Rail Logic Don't Add Up. In *CHES*, volume 4727 of *LNCS*, pages 95–106. Springer, 2007. Vienna, Austria. 1.3.3.1

[41] R. Soares, N. Calazans, V. Lomne, T. Ordas, P. Maurine, L. Torres, and M. Robert. Evaluation on FPGA of Triple Rail Logic Robustness against DPA and DEMA. In *DATE, track A4 (*Secure embedded implementations*)*. IEEE Computer Society, April 20–24 2009. Nice, France. 1.3.3.2

[42] François-Xavier Standaert, Siddika Berna Örs, Jean-Jacques Quisquater, and Bart Preneel. Power analysis attacks against FPGA implementations of the DES. In *Field Programmable Logic and Applications*, pages 84–94, London, UK, August 2004. Springer-Verlag. 4.1.1

[43] François-Xavier Standaert, Gaël Rouvroy, and Jean-Jacques Quisquater. FPGA Implementations of the DES and Triple-DES Masked Against Power Analysis Attacks. In *proceedings of FPL 2006*, August 2006. Madrid, Spain. 1.3.3.1

[44] François-Xavier Standaert, Siddika Berna Örs, Jean-Jacques Quisquater, and Bart Preneel. Power-Analysis Attacks Against FPGA Implementations of the DES. In *FPL 2004*, volume 3203, pages 84–94. (document)

[45] Daisuke Suzuki, Minoru Saeki, and Tetsuya Ichikawa. Random Switching Logic: A Countermeasure against DPA based on Transition Probability. 2004. http://eprint.iacr.org/2004/346. 1.3.3.1

[46] K. Tiri and I. Verbauwhede. A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation. In *DATE'04*, pages 246–251. IEEE Computer Society, February 2004. Paris, France. 1.3.3.2

[47] Kris Tiri. Side-Channel Attack Pitfalls. In *44th Design Automation Conference (DAC)*, pages 15–20, 2007. June 4 & 8, San Diego, California, USA. (document), 1.2.1.2, 7.2.2.1

[48] Kris Tiri, Moonmoon Akmal, and Ingrid Verbauwhede. A dynamic and differential cmos logic with signal independent power consumption to withstand differential power analysis on smart cards. pages 403–406, 2002. 1.3.3.2

[49] M. Ward. EMV card payments - An update. In *Information Security Technical Report*, volume 11 Issue 2, pages 89–92, 2006. http://www.sciencedirect.com/science/article/B6VJC-4K4H47T-5/1/62dfd637365ffa4aabb6dc205ab5f28f. (document)

[50] Siddika Berna Örs, Elisabeth Oswald, and Bart Preneel. Power-Analysis Attacks on an FPGA – First Experimental Results. In *CHES 2003*, volume 2779, pages 35–50. (document)

Bibliography: FPGA

[51] Victor Aken'Ova, Guy Lemieux, and Resve Saleh. An Improved Soft eFPGA Design and Implementation Strategy. In *IEEE Custom Integrated Circuits Conference*, 2005. 5.1

[52] Vaughn Betz and Jonathan Rose. Circuit Design, Transistor Sizing and Wire Layout of FPGA Interconnect. C-20:1469–1479, 1971. 5.3.1

[53] Vaughn Betz and Jonathan Rose. VPR: A New Packing, Placement and Routing Tool for FPGA Research. *Int'l Workshop on FPL*, pages 213–222, 1997. 2.3.4, 5.1, 6.2.1

[54] Vaughn Betz, Jonathan Rose, and Alexander Marquardt. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1999. 2.1.1, 2.3.1, 2.3.3, 5.3.1, 6.2.1, 7.4, 7.4.1

[55] P. K. Chan and M. D. F. Schlag. Acceleration of an fpga router. In *FCCM '97: Proceedings of the 5th IEEE Symposium on FPGA-Based Custom Computing Machines*, page 175, Washington, DC, USA, 1997. IEEE Computer Society. 6.10

[56] Yao-Wen Chang, D. F. Wong, and C. K. Wong. Universal switch modules for FPGA design. *ACM Trans. Des. Autom. Electron. Syst.*, 1(1):80–101, 1996. 2.2.1.2

[57] Hongyu Chen, Chung-Kuan Cheng, Andrew B. Kahng, Ion I. Mandoiu, Qinke Wang, and Bo Yao. The y architecture for on-chip interconnect: analysis and methodology. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 24(4):588–599, 2005. 2.4.2

[58] Phillip Christie and Dirk Stroobandt. The Interpretation and Application of Rent's Rule. *IEEE Transactions on VLSI Systems*, 8, NO.6, 2000. 2.4.1, 2.4.3.3

[59] Katherine Compton and Scott Hauck. Totem: Custom Reconfigurable Array Generation. In *IEEE Symposium on FPGAs for Custom Computing Machines*, 2001. 5.1

[60] André Dehon. Balancing interconnect and computation in a reconfigurable computing array. In *FPGA '99*, pages 69–78, New York, NY, USA, 1999. ACM Press. 2.4.1.5

[61] André Dehon. Unifying mesh and tree based programmable interconnect. *IEEE Transactions on VLSI Systems*, 12(10):1051 – 1065, 2004. 2.2.2, 2.4.1.5, 2.4.2

[62] William E. Donath. Placement and Average Interconnection Lengths of Computer Logic. *IEEE Transactions on Circuits and Systems*, CAS-26,NO-4:272 – 277, 1979. 2.4.1, 2.4.3.4, 2.4.3.4, 8.4

[63] B. Donlin and H. Singh. A Dynamic Reconfiguration Run Time System. *Proc. 5th Annual IEEE Symposium on Custom Computing Machines, IEEE Computer Society Press*, pages 66–75, 1997. 6.1

[64] Alfred .E . Dunlop and Brian .W. Kerninghan. A Procedure for Placement of Standard Cell VLSI Circuits. *IEEE Transactions on Computer Aided Design*, CAD-4 No.1, 1985. 2.4.1.4

[65] Lars W. Hagen, Andrew B. Kahng, Fadi J. Kurdahi, and Champaka Ramachandran. On the intrinsic rent parameter and spectra-based partitioning methodologies. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 13(1):27–37, 1994. 2.4.1.1, 2.4.1.3, 2.4.3.3

[66] Edson L. Horta, John W. Lockwood, David E. Taylor, and David Parlour. Dynamic Hardware Plugins in an FPGA with Partial Run-time Reconfiguration. In *Design Automation Conference (DAC)*, New Orleans, LA, June 2002. 6.1, 6.3

[67] Michael Hutton. Interconnect Prediction for Programmable Logic Devices. In *SLIP*, pages 125–131. ACM, 2001. Sonoma, California, United States. 6.7, 7.9

[68] Design Contest.
http://www.doc.ic.ac.uk/~pjamieso/FPT_DESIGN_08/. 2.1.1

[69] M. Igarashi, T. Mitsuhashi, A. Le, S. Kazi, Yang-Trung Lin, A. Fujimura, and S. Teig. A diagonal interconnect architecture and its application to RISC core design. *Solid-State Circuits Conference Digest of Technical Papers. ISSCC. 2002 IEEE International*, 2:166–167, 2002. 2.4.2

[70] Peter Jamieson and Jonathan Rose. Enhancing the area-efficiency of fpgas with hard circuits using shadow clusters. *Field Programmable Technology, 2006. FPT 2006. IEEE International Conference on*, pages 1–8, Dec. 2006. 2.1.1

[71] K.Appel, W.Haken, and J.Koch. Every Planar Map is Four Colorable. *Journal of Mathematics*, pages 491–567, 1977. 2.4.1.3, 2.4.2

[72] S .Kirkpatrick, C.D Gelatt Jr., and M.P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983. 2.3.3, 2.4.1.4

[73] I. Kuon, A. Egier, and J. Rose. Design, Layout and Verification of an FPGA using Automated Tools. In *ACM Symposium on FPGAs*, pages 215–226, Feb 2005. 5.1

[74] Ian Kuon, Aaron Egier, and Jonathan Rose. Design, layout and verification of an fpga using automated tools. In *FPGA '05: Proceedings of the 2005 ACM/SIGDA*, pages 215–226, New York, NY, USA, 2005. ACM. 5.3.4

[75] B.S Landman and R.L Russo. On a Pin Versus Block Relationship For Partitions of Logic Graphs. *IEEE Transactions on Computers*, C-20:1469–1479, 1971. 2.4.1, 2.4.3.3

[76] G. Lemieux, E. Lee, M. Tom, and A. Yu. Directional and single-driver wires in FPGA interconnect. *Field-Programmable Technology, 2004. Proceedings. 2004 IEEE International Conference on*, pages 41–48, Dec. 2004. 7.5

[77] David Lewis, Elias Ahmed, Gregg Baeckler, Vaughn Betz, Mark Bourgeault, David Cashman, David Galloway, Mike Hutton, Chris Lane, Andy Lee, Paul Leventis, Sandy Marquardt, Cameron McClintock, Ketan Padalia, Bruce Pedersen, Giles Powell, Boris Ratchev, Srinivas Reddy, Jay Schleicher, Kevin Stevens, Richard Yuan, Richard Cliff, and Jonathan Rose. The Stratix II logic and routing architecture. In *FPGA '05: Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, pages 14–20, New York, NY, USA, 2005. ACM. 2.1.1, 2.1.1, 2.1.1, 2.3.1

[78] Jason Luu, Ian Kuon, Peter Jamieson, Ted Campbell, Andy Ye, Wei Mark Fang, and Jonathan Rose. Vpr 5.0: Fpga cad and architecture exploration tools with single-driver routing, heterogeneity and process scaling. In *FPGA '09: Proceeding of the ACM/SIGDA international symposium on Field programmable gate arrays*, pages 133–142, New York, NY, USA, 2009. ACM. 7.5

[79] Alexander Marquardt, Vaughn Betz, and Jonathan Rose. Timing-Driven Placement for FPGAs. *Int'l Workshop on FPL*, pages 213–222, 1997. 6.3.2

[80] Zied Marrakchi, Hayder Mrabet, and Habib Mehrez. A new Multilevel Hierarchical MFPGA and its suitable configuration tools. *isvlsi*, 0:263–268, 2006. 2.2.2

[81] Larry Mcmurchie and Carl Ebeling. Pathfinder: A negotiation-based performance-driven router for fpgas. pages 111–117, 1995. 2.3.4

[82] D. Mesquita, B. Badrignans, L. Torres, G. Sassatelli, M. Robert, and F. Moraes. A cryptographic coarse grain reconfigurable architecture robust against dpa. pages 1–8, March 2007. 6.1

[83] Daniel Mesquita, Lionel Torres, Fernando Gehm Moraes, Gilles Sassatelli, and Michel Robert. Are coarse grain reconfigurable architectures suitable for cryptography? In *VLSI-SOC*, pages 276–281, 2003. 6.1

[84] K. Nasi, T. Karouhalis, M. Danek, and Z. Pohl. FIGARO – an automatic tool flow for designs with dynamic reconfiguration. *International Conference on Field Programmable Logic and Applications*, pages 590–593, Aug 24-26 2005. 6.1

[85] S. Phillips and S. Hauck. Automatic Layout of Domain Specific Reconfigurable Subsystems for System-on-Chip. In *FPGA*, 2002. 5.1

[86] Joachim Pistorius and Mike Hutton. Placement Rent Exponent Calculation Methods, Temporal Behaviour and FPGA architecture Evaluation. In *SLIP*, pages 31–38, April 5-6 2003. 2.4.1, 2.4.1.3, 2.4.1.5

[87] J. Rose and S. Brown. Flexibility of interconnection structures for field-programmable gate arrays. *Solid-State Circuits, IEEE Journal of*, 26(3):277–282, Mar 1991. 2.2.1.1, 2.2.1.1

[88] R.P.S. Sidhu, A. Mei, S. Wadhwa, and V.K. Prasanna. A self-reconfigurable gate array architecture. *10th International Workshop FPL 2000*, August 2000. 6.1

[89] E. Tau, I. Eslick, D. Chen, J. Brown, and A. DeHon. A first generation DPGA implementation. In *In Proceedings of the Third Canadian Workshop on Field-Programmable Devices*, May 1995. 6.1

[90] Grant Wigley and David Kearney. The Development of an Operating System for Reconfigurable Computing. In *IEEE Symposium on FPGAs for Custom Computing Machines, Napa Valley*, volume , 2001. 6.3.1

[91] Steven J.E. Wilton, Noha Kafafi, James C.H. Wu, Kimberly A. Bozman, Victor O.Aken'Ova, and Resve Saleh. Design considerations for soft embedded programmable logic cores. *IEEE Journal of Solid-State Circuits*, 40:485–496, February 2005. 2.2.1.3, 5.1, 6.2.2.2, 7.4.3

[92] Michael G. Wrighton and André M. DeHon. Hardware-assisted simulated annealing with application for fast fpga placement. In *FPGA '03: Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays*, pages 33–42, New York, NY, USA, 2003. ACM. 6.10

Bibliography: Async

[93] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D(3):315–325, 1997. 3, 3.6.2

[94] B. Gao. *A globally asynchronous locally synchronous configurable array architecture for algorithlm embeddings*. PhD thesis, University of Edinburgh, UK, December 1996. 4.2

[95] Marcel Vand Der Goot. *Semantics of VLSI Synthesis*. 1995. 3.2.1

[96] S. Hauck. Asynchronous Design Methodologies: an Overview. In *IEEE*, volume 83, pages 69–93, 1995. 6.1

[97] Scott Hauck, Gaetano Boriello, and Carl Ebeling. Montage: An FPGA fo Synchronous and Asynchronous Circuits. In *2nd International Workshop on Field-Programmable Logic and Applications*, Vienna, August 1992. 4.2

[98] C. A. R. Hoare. *Communicating sequential processes*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1985. 3.2.1

[99] N. Huot, H. Dubreuil, L. Fesquet, and M. Renaudin. FPGA Architecture for Multi-Style Asynchronous Logic. In *Design, Automation and Test in Europe (DATE'05)*, volume 1, pages 32–33, 2005. 6.1

[100] C.R. Jesshope, I.M. Nedelchev, and C.G. Huang. Compilation of process algebra expressions into delay-insensitive circuits. *Computers and Digital Techniques, IEE Proceedings E*, 140(5):261–268, Sep 1993. 3.4.2

[101] Daniel H. Linder and James C. Harden. Phased Logic Supporting the Synchronous Design Paradigm with Delay-Insensitive Circuitry. *IEEE Transactions on Computers*, 45(9):1031–1044, 1996. 3, 3.6.1

[102] Kapilan Makeswaran and Venkatesh Akella. PGA-STC : Programmable Gate Array for Implementating Self-Timed Circuits. *International Journal of Electronics*, 84:255–267, 1998. 4.2

[103] Rajit Manohar. A case for asynchronous computer architecture. In *ISCA Workshop on Complexity-Effective Design*, 2000. 3.2.1

[104] A.J. Martin and M Nystrom. Asynchronous techniques for system-on-chip design. *Proceedings of the IEEE*, pages 1089–1120, June 2006. 3.3.3

[105] Alain J. Martin. The Limitations to Delay-Insensitivity in Asynchronous Circuits. In *Sixth MIT Conference on Advanced Research in VLSI*, pages 263–278, 1990. 3, 3

[106] Yannick Monnet, Marc Renaudin, and Régis Leveugle. Designing Resistant Circuits against Malicious Faults Injection Using Asynchronous Logic. *IEEE Trans. Computers*, 55(9):1104–1115, 2006. 4.4

[107] S. Moore, R. Anderson, P. Cunningham, R. Mullins, and G. Taylor. Improving Smart Card Security using Self-timed Circuits. In *ASYNC'02*, pages 211–218, April 2002. Manchester, United Kingdom. 4.4

[108] D. E. Muller and W. S. Bartky. A theory of asynchronous circuits. In *International Symposium on the Theory of Switching, The Annals of the Computation Laboratory of Harvard University*, volume 29, pages 204–243, 1959. 3.5.1

[109] Tadao Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, 1989. 3.2.2, 3.2.2

[110] Robert Payne. *Self Timed Field Programmable Gate Array Architectures*. PhD thesis, University of Edinbugh, 1997. 4.2

[111] C.L. Seitz. System timing, Introduction to VLSI Systems. In *Addison-Wesley*, pages 218–262, 1980. 3

[112] I. E. Sutherland. Micropipelines. *Commun. ACM*, 32(6):720–738, 1989. 3

[113] John Teifel and Rajit Manohar. Programmable Asnchronous Pipeline Arrays. In *International Workshop on Field-Programmable Logic and Applications*, Lisbon, Portugal, September 2003. 4.2

[114] John Teifel and Rajit Manohar. An Asynchronous Dataflow FPGA Architecture. *IEEE Transactions on Computers*, 53(11):1376–1392, 2004. 4.2

[115] John Teifel and Rajit Manohar. Highly Pipelined Asynchronous FPGAs. In *12th ACM International Symposium on Field-Programmable Gate Arrays*, Monterey, CA, February 2004. 4.2

Bibliography: Misc

[116] DPA Contest. http://www.dpacontest.org/. 4.1

[117] M2000 FLEXEOStm Configurable IP Core. http://www.m2000.fr. 5.1

[118] MARS project website. http://www.comelec.enst.fr/recherche/mars/. 6.4.1

[119] Mentor Graphics. http://www.mentor.com/products/ic_nanometer_design/analog-mixed-signal-verification/eldo/. 7.6, 7.8.1

[120] Secured Asynchronous FPGA for Embedded systems. http://www.comelec.enst.fr/recherche/safe/. 6.1

[121] STARRCXT Synopsis. http://www.synopsys.com/TOOLS/IMPLEMENTATION/SIGNOFF/Pages/Star-RCXT.aspx. 7.8.1, III

[122] VariCore Embedded Programmable Gate Array Core(EPGA). http://www.actel.com. 5.1

[123] Virtual Socket Interface Alliance – VCI Standard. http://www.vsia.org/. 6.2.1

[124] ALTERA. Benchmark designs for the quartus university interface program (quip) version 1.0, January 2006. 7.10

[125] Atmel. Programmable Logic and microcontroller products, 2005. http://www.atmel.com/. 6.1

[126] Axis. Fox board, 2005. http://www.acmesystems.it/?id=4. 6.6.1

[127] J.R. Black. Electromigration failure modes in aluminum metallization for semiconductor devices. *Proceedings of the IEEE*, 57(9):1587–1594, Sept. 1969. 8.1

[128] L. Chua. Memristor-the missing circuit element. *Circuit Theory, IEEE Transactions on*, 18(5):507–519, Sep 1971. 8.3

[129] F Dabiri and M Potkonjak. Hardware aging-based software metering. In *DATE 09*, Nice, France, April 20-24 2009. 8.2

[130] W. Wayt Gibbs. How Hackers Can Steal Secrets from Reflections. *Scientific American*, May 2009. (document)

[131] Puneet Gupta and Andrew B. Kahng. Bounded-lifetime integrated circuits. In *DAC '08: Proceedings of the 45th annual conference on Design automation*, pages 347–348, New York, NY, USA, 2008. ACM. (document), 8, 8.2, 8.2

[132] Maxima. www.gnu.org/software/maxima/maxima.html. 8.4

[133] Michael John Sebastian Smith. Application-Specific Integrated Circuits. In *Addison-Wesley*, 1997. 7.2.1.1

[134] Xilinx. *Development System Reference Guide*. http://toolbox.xilinx.com/, 2005. 6.1

Publications

[135] Taha Beyrouthy, Alin Razafindraibe, Laurent Fesquet, Marc Renaudin, Sumanta Chaudhuri, Sylvain Guilley, Philippe Hoogvorst, and Jean-Luc Danger. A Novel Asynchronous e-FPGA Architecture for Security Applications. In *ICFPT*, pages 15–22, 2007. Kokurakita, Kitakyushu, JAPAN. (document), 7.3.0.4

[136] S. Chaudhuri, J-L. Danger, P. Hoogvorst, and S. Guilley. Efficient Tiling Patterns for Reconfigurable Gate Arrays (Extended Abstract in FPGA 2008). In *SLIP*, pages 11–18, Newcastle, UK, April 2008. ACM. (document)

[137] Sumanta Chaudhuri, Jean-Luc Danger, and Sylvain Guilley. Efficient Modeling and Floorplanning of Embedded-FPGA Fabric. In *FPL*, pages 665–669. IEEE, Aug 2007. Amsterdam, Netherlands. (document), 5.3.3, 7.7.2

[138] Sumanta Chaudhuri, Jean-Luc Danger, Sylvain Guilley, and Philippe Hoogvorst. FASE: An Open Run-Time Reconfigurable FPGA Architecture for Tamper-Resistant and Secure Embedded Systems. In *3rd international conference on reconfigurable computing and FPGAs (ReConFig 2006)*, September 2006. (document)

[139] Sumanta Chaudhuri, Sylvain Guilley, Florent Flament, Philippe Hoogvorst, and Jean-Luc Danger. An 8x8 Run-Time Reconfigurable FPGA Embedded in a SoC. In *DAC*, Anaheim, California USA, June 2008. ACM/IEEE. (document)

[140] Sumanta Chaudhuri, Sylvain Guilley, Philippe Hoogvorst, Jean-Luc Danger, Taha Beyrouthy, Alin Razafindraibe, Laurent Fesquet, and Marc Renaudin. An Asynchronous FPGA Architecture for Cryptographic Applications **(in Submission)**. In *ACM Transactions on Reconfigurable Technology and Systems*, 2008. (document)

[141] Sumanta Chaudhuri, Sylvain Guilley, Philippe Hoogvorst, Jean-Luc Danger, Taha Beyrouthy, Alin Razafindraibe, Laurent Fesquet, and Marc Renaudin. Physical Design of FPGA Interconnect to Prevent Information Leakage. In *ARC (Applied Reconfigurable Computing), Proceedings in LNCS Springer-Verlag Berlin Heidelberg*, volume 4943, pages 87–98, London, UK, mar 2008. (document), 7.4.7

[142] Sylvain Guilley, Laurent Sauvage, Philippe Hoogvorst, Renaud Pacalet, Guido Marco Bertoni, and Sumanta Chaudhuri. Security evaluation of wddl and seclib countermeasures against power attacks. *IEEE Transactions on Computers*, 57(11):1482–1497, 2008. (document), 1.3.3.2, 4.2

[143] Philippe Hoogvorst, Sylvain Guilley, Sumanta Chaudhuri, Alin Razafindraibe, Taha Beyrouthy, and Laurent Fesquet. A Reconfigurable Cell for a Multi-Style Asynchronous FPGA. In *ReCoSoC*, pages 15–22, 2007. (document), 7.3

Index