Hardware implementation of cell-inspired computational models

Zeyi Shang

UNIVERSITÉ PARIS-EST CRÉTEIL VAL DE MARNE

DOCTORAL THESIS

Hardware implementation of cell-inspired computational models

---

Author: Zeyi Shang
Supervisor: Gexiang Zhang
Jury President: Lynda Mokdad
Sergey Verlan

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy

in

COMPUTER SCIENCE

Declaration of Authorship

I, Zeyi SHANG, declare that this thesis titled, "Hardware implementation of cell-inspired computational models" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Signed:

Date:

"L'archer à cheval de la steppe a régné sur l'Eurasie pendant treize siècles parce qu'il était la création spontanée du sol même, le fils de la faim et de la misère, le seul moyen pour les nomades de ne pas entièrement périr les années de disette."

René Grousset

L'empire des steppes

Abstract

Zeyi SHANG

Hardware implementation of cell-inspired computational models

Parallelism, non-determinism and large scale are the three characters of biological and ecological systems which are transmitted to bio-inspired computing models enlightened by these bio- or/and eco- systems. Imitating bio-inspired computing models on general purpose computers by designing high level programming language codes is the common approach to simulate these unconventional computing models for its accessibility. However, this approach is inappropriate to cope with the three characters mentioned above, especially for the parallelism on a large scale. From the perspective of hardware, CPU of computers executes software codes which simulate bio-inspired computing models. More precisely, integrated circuits inside CPUs or other processing devices perform operations defined by software codes which mimic these models. Software codes emulating bio-inspired computing models can get expected results, but it is not guaranteed that the CPU executes operations in line with what models do. If the parallel performance provided by target CPU is not enough to support parallel processing, some parallel procedures will be serialized. And because that the process of CPU is not transparent, we cannot know whether the CPU carry out operations in accordance with that of models or not. To state the main works more clearly, the notion of implementation and simulation should be distinguished. If a hardware emulates bio-inspired computing models in strict accordance with procedures defined by the models, such type of emulation is termed as "implementation". While if processing procedures of target hardware are not consistent with models, although expected outcomes obtained, such kind of emulation is entitled as "simulation".

A more preferable approach to imitate bio-inspired computing models is to design model-specific hardware platform to implement other than simulate them for the simulation will do harm for some advantages of these models, for instance, the large scale parallelism. The hardware implementation method of bio-inspired computing models imitates models with (digital) circuits directly instead of software codes since circuits are the principal part comprising a variety of processors such as CPUs and GPUs. On the contrary, if a processor is powerful enough to execute models as exactly the theoretical procedures, this emulation is called software implementation. High-end GPUs with computing unified device architecture (CUDA) belong to such type of hardware so many parallel algorithms are implemented on them. This thesis concentrate on hardware implementation of two bio-inspired computing models: reaction systems and numerical P systems. Variants of these two models were implemented on the reconfigurable hardware—field programmable gate array (FPGA). The prominent strength of hardware implementation comparing with software implementation is its high performance rooted in the tailored parallel hardware architectures.

Reaction systems is a theoretical framework for investigating biochemical reactions and their interactions in biological cells. It is a qualitative parallel computing model for reactions and their interactions taking place concurrently. There are investigations on CPU software simulation and GPU software implementation of reaction systems. For the simulation of large-scale reaction systems, the processing speed of these two methods is not adequate. Hence, it is worthwhile to investigate their implementation using hardware circuits. To conceive the FPGA implementation method of reaction systems, the relationship between reaction systems and synchronous circuits is studied at first. It is found that the dynamics of a reaction system corresponds to the dynamics of a Mealy or Moore automaton of some synchronous circuit, and the qualitative characteristics of reaction system can be expressed by the value of binary variables. Based on these observations, an FPGA implementation method for reaction systems is devised. According to this method, the reaction system of intermediate filaments self-assembly and heat shock response reaction system are implemented on FPGA, which verify the correctness of this method. The calculation speed is up to 2×10^8 step per second (the frequency is 200 MHz). A binary counter constructed with a reaction system is implemented on FPGA as well, exhibiting the expressibility of reaction systems that can be used to build a quantitative model.

Membrane computing is a parallel quantitative computing paradigm inspired by the membrane structure of biological cells and biochemical reactions inside. Membrane computing models are called P systems. Due to the limited parallel processing capability of CPU, software simulation of P systems cannot leverage their maximal parallelism. Like reaction systems, implementing P systems on parallel hardware circuits is more reasonable. When hardware resources are sufficient, theoretical parallelisms can be fully exploited. In this thesis, FPGA implementations of the cell-like symbol object P systems are summarized, and existing FPGA implementation methods are expounded at length. The qualitative and quantitative comparisons of various methods are formulated. Object distribution problem (ODP, choosing which rules to apply) is the core problem of hardware implementation of symbol object P systems. Because of the maximal parallelism and non-determinism of this class of P system, ODP is NP hard. In order to calculate the solutions (available rule sets) of ODP, a method based on multi-criteria optimization and integer linear programming is proposed.

This thesis also considers on a special type of cell-like P system, called numerical P system (NPS), focusing on its FPGA implementation and applications in robot control and path planning. The relationship between NPS and system of difference equations is studied, and it is found that NPS is another form of the latter. Considering that a difference equation and a differential equation can be transformed to each other, NPS can model systems modeled by differential equations. Enzymatic numerical P system (ENPS) is a variant of NPS having better modeling capacity for the function of enzyme-like variables. However, the rule usability predicate form is not uniform, and there is only one discriminant, which limits its application scope. In order to expand the application field of (E)NPS and facilitate FPGA implementation, they are extended to generalized numerical P system (GNPS). The rule applicability discriminants of GNPS are generalized to Presburger arithmetic.

The FPGA implementation method of GNPS's rules, called *FPGA step-wise parallel implementation of program* (FSPIP) implementation method, is proposed. According to this method, the robot membrane controller based on NPS, ENPS and GNPS are implemented. Since GNPS is a superset of (E)NPS, the FSPIP approach also compatible to (E)NPS. Compared with software simulation, the acceleration ratio

obtained by the three NPSs implemented on FPGA is the order of 10^4. Considering the data input and output in practical applications, an UART serial communication module was designed for the NPS membrane controller to receive data from the robot sensors and to transmit the computation results to actuators. A object multiset rewriting rule of symbol object P system can be disassembled as a set of NPS programs. Programs updating multiplicity of the same object can be combined together. The function of combined programs are equivalent to rewriting rules in the same membrane.

The maximal parallelism of GNPS is suitable to accelerate computationally intensive algorithms. The rapidly-exploring random tree (RRT) algorithm is modeled in the framework of GNPS, obtaining a RRT-GNPS model in which all parallel operations are calculated simultaneously, while sequential procedures are processed serially. The RRT-GNPS model is implemented on FPGA in line with the GNPS FPGA implementation method. In consideration of future applications in large-scale maps, variables in RRT-GNPS are expressed in IEEE 754 single-precision floating point format, which is different from fixed-point number representation used in the implementation of membrane controllers. Floating-point adder, multiplier and reverse square root units are designed. A sequential triggering method based on edge detection is designed to trigger operations according to the applicability of rules and the order of arithmetic operation priority. Compared with software simulation, the FPGA hardened RRT-GNPS achieves a speedup of 3.20195×10^4.

Résumé

Le non-déterminisme et le parallélisme à grande échelle sont deux propriétés importantes des systèmes biologiques et écologiques. Il est donc naturel qu'elles soient héritées par les modèles de calculs bio-inspirés, c.-à-d. dont le fonctionnement est inspiré par des processus biologiques. Pour utiliser ces modèles en pratique, ou juste pour mieux comprendre leur comportement, il est important de pouvoir simuler leur fonctionnement. L'approche standard pour le faire consiste dans la réalisation des simulateurs logiciels, exécutées sur des ordinateurs conventionnels. Cependant, l'architecture de ces derniers n'est pas de tout compatible avec les propriétés essentielles des systèmes visés, plus particulièrement avec le parallélisme à grande échelle. Même si le résultat final est correct, l'exécution des modèles ne corresponde pas à leur inspiration biologique et leur définition mathématique, la plupart des opérations parallèles étant effectuées en séquence. On parle alors de simulation, qui traite le modèle considéré comme une boite noire en fournissant une entrée et en produisant la bonne sortie après un certain temps. Cette simulation peut être séquentielle (sur un processeur), ou parallèle (sur un processeur multicœur ou une carte graphique ayant plusieurs processeurs spécialisés). En revanche, l'implémentation (ou la simulation forte), demande additionnellement que le comportement du modèle original soit respecté lors de l'exécution.

Pour obtenir une implémentation d'un modèle de calcul bio-inspiré il est possible d'utiliser des composants biologiques, directement liés au phénomène dont le modèles est inspiré. Cette approche, réalisable dans certains cas, est bornée par les limites actuelles de la biotechnologie. Une autre possibilité repose sur le parallélisme inhérent des circuits logiques. Il est possible de construire une plateforme matérielle in silico dont les circuits vont correspondre aux composants du modèle tout en gardant le fonctionnement parallèle à grande échelle. Pour faciliter le développement on utilise généralement des circuits logiques matériels reconfigurables, appelés FPGA, qui permettent un prototypage rapide des circuits matériels.

Cette thèse s'articule autour de l'implémentation matérielle de deux modèles de calcul bio-inspirés : systèmes à réaction (RS) et des systèmes à membranes numériques (NPS). Nous avons implémenté différentes versions de ces modèles sur FPGA et nous avons comparé les résultats obtenus avec les simulations logicielles et matérielles (en utilisant les GPUs) existantes. Afin d'avoir des exemples non-triviaux, nous avons également proposé la traduction des différents problémes liées au contrôle du mouvement robotique dans le formalisme NPS.

Les systèmes à réaction sont une formalisation théorique des réactions biochimiques. C'est un modèle de calcul qualitatif et parallèle qui se concentre sur les relations causales entre les entités en ignorant les quantités (ou les concentrations). La dynamique du modèle repose sur la promotion et l'inhibition d'un réactant par des autres. Le principe de non-permanence assure la présence des réactants dans le système uniquement s'ils viennent d'être produits ou introduits. Il existe plusieurs logiciels de simulation des systèmes à réaction, ainsi que des simulations matérielles à l'aide des GPU. Cependant, aucune implémentation matérielle n'a pas été reporté dans la littérature. Pour des circuits de petite taille cela ne pose aucun problème

particulier, car les temps de simulation logicielle sont suffisamment petits. Par contre, le passage à l'échelle et l'utilisation des modèles de grande taille nécessite une approche différente.

Dans cette thèse on s'est intéressé à l'implémentation matérielle des systèmes à réaction à l'aide des FPGA. Premièrement, on a étudié le lien entre les systèmes à réaction et les circuits de commutation synchrones (qui sont un sous-ensemble des circuits Booléens), ainsi que les modèles connexes des automates de Mealy et Moore (qui sont des automates finis avec des entrées/sorties). Nous avons trouvé que les deux modèles sont très proches et nous avons donné des algorithmes de traduction entre les systèmes à réaction et les circuits de commutation synchrones qui préservent la même dynamique pas à pas. Cela a permis par le passage de trouver une forme normale pour les systèmes à réaction ayant des propriétés intéressantes et de proposer de nouvelles possibilités de conception de ces derniers à l'aide des outils bien connus comme des automates de Mealy.

Les circuits de commutation peuvent être traduits naturellement vers FPGA. Cela permet donc de traduire un système à réaction vers un circuit FPGA tout en gardant son comportement dynamique pas à pas, en obtenant donc une implémentation. Pour tester notre approche nous avons implémenté des systèmes à réaction proposés pour la modélisation de la croissance des filaments intermédiaires et de la réponse au choc thermique dans les bactéries. Ensuite, nous avons développé un compilateur qui permet d'automatiser la traduction en appliquant les algorithmes proposés. Cela nous a permis d'effectuer des mesures de performance de notre approche sur des systèmes de grande taille: nous avons effectué l'implémentation du modèle du transduction du signal du récepteur ErbB dans les cellules épithéliales humaines qui comprend 620 réactions et 246 entités. Cette implémentation a permis d'atteindre la vitesse de calcul de 108 étapes/seconde et d'avoir une accélération de l'ordre 2.5×10^5 par rapport aux meilleurs simulateurs GPU existants.

Les autres modèles de calcul considérés dans ce mémoire relèvent du domaine du calcul à membranes qui est un paradigme de calcul parallèle et quantitatif inspiré par la structure et le fonctionnement des cellules vivantes. Le fonctionnement du modèle sous-entend un parallélisme massif, sa simulation sur des ordinateurs conventionnels ou même GPU ne permet pas de tirer profit de cette propriété. En revanche, une implémentation à l'aide des FPGA permet d'obtenir un fonctionnement plus proche de la sémantique du modèle et d'avoir par conséquent un gain de vitesse.

Dans ce mémoire de thèse nous donnons un aperçu des méthodes d'implémentation existantes des différentes variantes des systèmes à membranes à l'aide des FPGA. Afin de pouvoir comparer les approches, nous avons représenté le problème calculatoire principal de ces implémentations (le calcul des règles applicables pour une configuration donnée) sous forme d'un problème d'optimisation multicritères. Nous avons montré que les différentes solutions existantes correspondent à des scalarisations particulières du problème d'optimisation.

La suite du mémoire se concentre sur une variante particulière des systèmes à membranes, appelée systèmes à membranes numériques (NPS). C'est un modèle qui comprend plusieurs compartiments chacun possédant des variables numériques. Les règles du modèle sont des fonctions qui calculent une valeur numérique à partir des variables présentes dans un compartiment, ainsi que dans les compartiments voisins et puis la distribuent vers d'autres variables en utilisant des pondérations. Le modèle NPS et son extension ENPS, ou l'applicabilité des règles est contrôlée par une valeur seuil, ont des applications importantes dans le contrôle robotique et la planification du chemin.

Nous avons montré des liens inattendus entre les NPS et les systèmes d'équations en différences finies en montrant comment transformer les modèles l'un dans l'autre sans perdre la dynamique. Plus précisément, la relation entre les NPS et les systèmes d'équations des différences finies est de même nature que celle entre les systèmes à réaction et les circuits de commutation. De plus, les systèmes à réaction peuvent être vues comme des versions simplifiées de NPS ou les variables ont des valeurs booléennes (à la place des valeurs numériques). Cela nous a conduit naturellement vers une nouvelle notion d'automates de Mealy opérant sur des vecteurs de nombres en entrée/sortie.

Nous avons également proposé une nouvelle extension du modèle (E)NPS, appelée GNPS, qui a pour but l'optimisation du codage des problèmes de contrôle robotique et qui en même temps admet une implémentation efficace à l'aide des FPGA. En effet, dans GNPS l'application des règles est contrôlée par des prédicats arithmétiques écrits en arithmétique de Presburger (avec une signature optionnelle). Cela permet à la fois de faciliter l'écriture des contrôleurs et d'avoir une implémentation directe sur FPGA. En se fondant sur les résultats théoriques ci-dessus, nous avons conçu une méthode d'implémentation des GNPS à l'aide des FPGA. Cette implémentation peut être faite avec deux représentations différentes des nombres: en point fixe et en format IEEE 754. Dans le premier cas des accélérations importantes ont été obtenues, car les opérations d'addition, soustraction, multiplication par constante et test sont facilement représentables en FPGA. Nous avons développé un contrôleur robotique à l'aide de GNPS et nous l'avons implémenté sur FPGA en utilisant la méthode proposée. En comparant avec une simulation sur GPU, notre implémentation a une accélération de l'ordre 10^4. Nous avons également rajouté un module de communication UART pour récupérer les données des capteurs et de contrôler le mouvement des roues en temps réel.

Comme le modèle GNPS est par définition parallèle et comme son implémentation en FPGA préserve cette propriété, il devient intéressant de l'utiliser pour la résolution des algorithmes demandant beaucoup de calculs. Nous avons modélisé l'algorithme RRT pour la planification du mouvement. Cela a demandé également de concevoir des modules de calcul des diverses fonctions arithmétiques utilisant des nombres encodés au format IEEE 754. D'après les tests, l'accélération obtenue est de l'ordre 10^4.

Organisation du mémoire :

Chapitre 1 contient l'introduction, ainsi que la description des notions principales utilisés dans la thèse.

Chapitre 2 donne les définitions des modèles considérés (systèmes à réaction, systèmes à membranes et systèmes à membranes numériques). Ce chapitre présente également comment on peut représenter une étape de calcul à l'aide d'un problème d'optimisation multicritères.

Chapitre 3 contient une introduction aux GPU et CUDA et présente un aperçu des implémentations matérielles existantes dans le domaine.

Chapitre 4 étudie les liens entre les systèmes à réaction et les circuits de commutation et présente l'implémentation des systèmes à réaction sur FPGA.

Chapitre 5 analyse les liens entre NPS et les systèmes d'équations des différences finies. Le modèle GNPS y est également introduit.

Chapitre 6 présente l'implémentation de 3 contrôleurs de mouvement robotique en utilisant GNPS et les méthodes développées.

Chapitre 7 présente une implémentation de l'algorithme RRT de planification du mouvement d'un robot. L'encodage des valeurs numériques utilise le standard IEEE

754. Plusieurs modules FPGA ont été crées pour gérer les opérations non-linéaires, comme par exemple la racine carrée.

Chapitre 8 présente les conclusions.

Acknowledgements

First and foremost, I would like to express my special thanks to my two supervisors: professor Zhang Gexiang and Sergey Verlan. The research subject of my doctoral phase is a new field for me. Professor Zhang chose the research direction for me based on my academic experiences. Due to the insufficient accumulation of relevant knowledge, I encountered many problems and difficulties in the early stage of my research. Professor Zhang answered questions and solved problems I raised. To provide more chances for me to learn cutting-edge knowledge, Professor Zhang contacted several high-level international scholars to give guidance to me. He helped me to get the chance to be a joint supervising PhD student. Professor Zhang provided very good hardware equipment which I need in my research, which ensured that the project progress went with a swing. In the monthly academic summary meeting, Professor Zhang suggested many forward-looking opinions and constructive guidance for my research and guided me to overcome difficulties. Professor Zhang's rigorous scientific research attitude deeply influenced me, which made me adapt to the high-intensity scientific research work gradually. Professor Zhang also gave me considerate care in my ordinary life. Because of my dietary custom, Professor Zhang always prepare extra dishes for me at every dinner party, so that I could eat well. With his words and deeds, Professor Zhang taught me how to conduct scientific research and how to face the difficulties in life and work positively. Under the guidance of Professor Zhang, I am able to set sail in my academic career.

Sincere gratitude from the bottom of my heart is also given to my supervisor Sergey Verlan for all things you have done to me. I would never forget that it is you who helped me involving in the joint supervision project, enrolling in Université Paris-Est and gaining scholarship so that I can study in Créteil nearly one year. Bio-inspired computing and FPGA development were totally new to me and I was completely at a loss what to do at beginning. From studying reaction system and its Verilog design, you guide me in the scene. Almost every week in Créteil you resolve problems raised by me, giving the detailed derivations so that I can understand entirely. You help me renting a studio, opening a bank account and starting living in France. To pick me up in the Charles de Gaulle airport at 6:30 am, you get up very early. You have no idea how moved I was to see you in the airport lobby. Under your guidance, I learn how to do research and how to write academic papers. I am deeply grateful to you for these things. I am quite lucky to meet you and it is my honor to be your student.

It behaves me to express my special thanks to Professor Gheorghe Păun. In the autumn of Wuhan, 2016, it was my first time to meet a member of European Academy of Science coming from Romania. I was excited and nervous to shake hands with you. After 5 days of listening your lectures about formal languages and P systems, I got some fundamental concepts in the membrane computing field. I bought a Chinese version of your monograph *Membrane Computing: An Introduction*. I would like your autograph but I dare not to walk to you. At last I got the strength to walk to the podium, saying what I want. You agreed nicely and wrote on the flyleaf "Happy computing–Gheorghe Păun". Your words encourage me to get through

adversities in front of me. The book with your hand-writing is a treasure for me.

To Ignacio, you are such a kindhearted man. I would never forget the hotpot you treat me in Seville. It is you who take me to visit the magnificent Royal Alcázar, and to look around the lovely city. You picked me up in the airport when I came to Seville to attend the brainstorming week of Membrane Computing and drove me to the airport again in the early morning. You explained me how the RRT algorithm works so that I can implemented it on FPGA. Thank you so much for your kindness and consideration for me.

To Junior, you are a gentleman with heartiness. You introduce me to Xiangxiang, who is a caring girl helping me a lot to make a living in Créteil. You help to get the gym card of our university so I have a place to do physical exercise so to keep healthy. To Xiangxiang, although we never ever meet, you were the only Chinese people I can talk to at the first time I came to France. You warned me to stay alert in subway and told where to find a room close to our university with good price. Thank your goodness and wish happiness for you and your family. To Steve, you do a great favor for me to get my long term pass card in the prefecture hall. I remember that it was a cold winter morning you came to help me for my french is too poor to handle this. In fact during that time you did not know me well, but you still got up and braved the cold to support me. Your kindness touches my heart all the time. I thank Paul for his heart-warming smile every time entering our working room. I enjoy the conversations about my home town and academic discussions with you. Thank your help to deal with my inscription. To Tau, a enthusiastic and generous man. Your jasmine tee and other scented tee are always on time in the morning which fresh my mind, reminding me the ancient tradition of east Asia. To Alex, a cordial man. I enjoy the academic discussions with you. Thank you, Paul and Tau to hold the farewell party by the lake of Créteil for me. You guys bought cakes, snacks and beers. We talked a lot over beer. It is a good memory for me.

To Wenya, your culinary skill is very good. I miss chives pancakes and steam breads you made. They are so delicious. I am appreciative to be a house mate of you. Wish you all the best. To Mengmeng, thank your benevolence and patience to listen to my grievances about life. The Pixian broad bean paste you bought from Chinese supermarket for me saved my life. I am grateful to know you. Wish you a bright future.

I am grateful to Université Paris-Est for providing a good laptop computer to me. It is quite helpful to do FPGA developments because post implementation timing simulation requires very good CPU performance and large RAM.

I express my appreciation to my wife, Yongli, for her support, dedication and love. During my worst time you give me warm family, a cute baby. You bring up our child while I work in Chengdu far away from you. I owe you a lot. Many thanks to my sister in law, Yongmei, for your support and consideration.

Last but not certainly not least, thank my grand parents and my parents who raised and cultivated me. My grandmother was the one who love me most in the world. I grew up with her endless love which helps me struggling through the hardest time. My mother scratch a living to devote the energies of her lifetime to my education. My younger brother experienced all the difficulties and happiness with me. Because the love and support from all of you, I gain enough courage to fight against fate.

Contents

Declaration of Authorship **iii**

Abstract **vii**

Résumé **xi**

Acknowledgements **xv**

Contents **xvii**

List of Figures **xxi**

List of Tables **xxvii**

1 Introduction **1**
1.1 Biological cell: foundation of life 1
1.2 Biological system modelling 2
1.3 Reaction systems 3
1.4 Membrane computing 4
1.5 FPGA based reconfigurable computing 5
1.6 Thesis statement 6
1.6.1 Main research contents 6
1.6.2 Contributions 8
1.7 Thesis organization 10

2 Definitions **13**
2.1 Reaction systems 13
2.2 P systems 14
2.2.1 Symbol object cell-like P system 14
2.2.2 Network of cells 17
2.3 Reduction the computation of P systems to multi-criteria optimization 20
2.3.1 Preliminaries 20
2.3.2 Rule choice as integer multi-criteria linear optimization problem 21
2.3.3 Tentative solutions 22
2.4 Wrap-up 26

3 Overview of hardware implementation of reaction systems and P systems **27**
3.1 GPU presentation 28
3.2 FPGA presentation 30
3.2.1 FPGA architecture 30
3.2.2 FPGA development flow 34
3.3 Software simulation and implementation of reaction systems: an overview 36
3.3.1 CPU simulation of reaction systems 36
3.3.2 GPU implementation of reaction systems 37

3.4 Background of hardware implementations of P systems 38
3.5 General ideas about hardware implementations of P systems 39
3.5.1 Data organization 40
3.5.2 Object Distribution Problem and Non-Determinism 40
3.6 Literature review of existing P system FPGA implementations 42
3.6.1 Region-based implementation 42
Petreska and Teuscher implementation 42
Nguyen simulation 44
3.6.2 Rule-based implementations 45
Nguyen implementation 46
Verlan and Quiros implementation 48
3.7 Micro-controller based implementation of P systems 50
3.8 Wrap-up 53

4 FPGA implementation of reaction systems 55
4.1 Relations between reaction systems and synchronous circuits 55
4.1.1 From reaction systems to switching circuits 57
4.1.2 From switching circuits to reaction systems 59
4.1.3 General ideas for FPGA implementation of reaction systems . . 61
4.2 FPGA implementation of self-assembly intermediate filaments reaction systems 62
4.3 FPGA implementation of heat shock response reaction systems 67
4.4 FPGA implementation of reaction system binary counter 69
4.4.1 Reaction system binary counter design 69
4.4.2 UART-RS counter design and implementation 72
4.5 Wrap-up 75

5 Theoretical investigations of numerical P systems 79
5.1 Numerical P system and enzymatic numerical P system 80
5.2 The relationship between (E)NPS and system of difference equations . 82
5.2.1 From (E)NPS to system of difference equations 82
5.2.2 From system of difference equations to (E)NPS 83
5.3 Binary and unary normal form of (E)NPS 84
5.4 The relations between symbol-object P system and NPS 86
5.5 Generalized Numerical P Systems 88
5.6 FPGA implementation of GNPS 91
5.7 Case studies 94
5.7.1 Case study 1 94
5.7.2 Case study 2 96
5.8 Wrap-up 100

6 FPGA implementation of robot membrane controller 101
6.1 FPGA implementation of membrane controller based on NPS 102
6.2 FPGA implementation of membrane controller based on ENPS 110
6.3 FPGA implementation of membrane controller based on GNPS 112
6.4 UART communication of NPS 116
6.5 Wrap-up 119

7 FPGA Architecture for Generalized Numerical P System modeled Rapidly-exploring Random Tree Algorithm 121
7.1 Rapidly-exploring random tree algorithm . . . 121
7.2 Floating point arithmetic units design . . . 125
7.2.1 Basics of IEEE 754 single precision floating point standard . . . 125
7.2.2 Sequential triggered IEEE 754 compliant adder . . . 128
7.2.3 Sequential triggered IEEE 754 compliant multiplier and FP comparator . . . 129
7.2.4 Inverse square root unit . . . 131
7.2.5 Floating point random number . . . 132
7.3 GNPS arranged RRT register transfer level model design . . . 132
7.4 FPGA implementation of RRT-GNPS . . . 135
7.5 Wrap-up . . . 137

8 Conclusions 141

A Associated figures of reaction systems 145

B Associated figures of GNPS 149

Bibliography 155

Publications by the author 179

List of Figures

1.1 Biological system or process modelling and biological computing. The comprehensive investigation of biological systems or processes is enlightening for theoretical computer scientists as well, resulting in biological computing a cross-discipline filed. 3
1.2 Realms involved in the investigations of this thesis. Reaction systems and P systems are tightly relevant while several models of them will be implemented on FPGAs as reconfigurable computing instances. . . 7
1.3 Relations of main research topics. 9

2.1 An example of a P system with two membranes. Membrane 1 contains a object multiset $a^5b^8c^6$ and four rules. Membrane 2 encompasses a object multiset $b^8c^11e^9$ and three rules. 15
2.2 A P system at its initial configuration. The subscript in_3 of obm_r in rule r_{11} means obm_r will be sent to membrane 3, so do rule r_{14}, r_{33}. While subscript out of obm_r in rule r_{21} signifies its obm_r will be sent out to environment. This is a static P system that there is no evolution rule which can dissolving or creating membranes. 17
2.3 All the feasible rule multisets of each membrane. The selected applicable rule multisets are highlighted in red. 18
2.4 Rules in red are those constituting the applied rule multiset. The exponents above rules denote thier instances in the rules multiset. Residuary object multisets after the application of rule multisets are shown in blue. 18
2.5 Feasible and Pareto-optimal solutions from Example 2.13. 21

3.1 (left) CUDA execution model, (right) CUDA memory model [110]. . . 29
3.2 A SM is composed of an array of SPs, shared memory and a couple of caches. A Multiple-Thread tissue module contains a SM instruction scheduler which conducts instruction flows [112]. 29
3.3 A 6T CMOS SRAM cell [115]. 30
3.4 A slice containing 2 logic cells. These 2 logical cells perform $y_1 = a \wedge (\bar{b} \vee c)$ and $y_2 = a \vee \bar{b} \vee \bar{c}$ respectively. 32
3.5 The interconnection of 5 slices. Connection blocks (CBs) and switching blocks (SBs) impart highly flexibility for FPGAs to reconfigure their architectures to perform various functions. The CB and SB can contain much more intersection points than that depicted in this figure. 33
3.6 Vivado FPGA development flow. 36

4.1 The Mealy automaton for the circuit described by Equation (4.2). The label of the state corresponds to the value of the vector (q_1, q_2). The label of the transition corresponds to the value of the input variable x and output y. 57

4.2 The Moore automaton for the circuit described by Equation (4.2). The label of the state corresponds to the value of the vector $(q_1, q_2)/y$. The label of the transition corresponds to the value of the input variable x. . 58
4.3 Moore machine for Example 4.6. The state label corresponds to the values of the vector (F, H, O) and the transitions are labeled by the value of T. The output is the label of the state. 59
4.4 Mealy machine for the 1101 sequence detector. It outputs 1 when the corresponding sequence is encountered as input. The state label corresponds to the vector (q_2, q_1). 60
4.5 Waveform comparison of interactive process of $\mathcal{R}_{ifl1} = (B_1, A_1)$. 64
4.6 Waveform comparison of interactive process of $\mathcal{R}_{ifl2} = (B_2, A_2)$. 65
4.7 Physical constraints settings of two intermediate filaments reaction systems. Gray circles with wide orange bars are pins allocated to RTL model ports. 66
4.8 Post implementation timing simulation of $\mathcal{R}_{ifl1}$. 67
4.9 Hardware debug of two intermediate filaments RSs. 68
4.10 Hardware resources dissipation and power consumption of $\mathcal{R}_{ifl1}$. . . . 68
4.11 Waveform comparison of interactive process 1 of $\mathcal{R}_{hsr} = (B_3, A_3)$. . . . 70
4.12 Hardware debug of two heat shock response RSs. 71
4.13 UART data frame with 8 bits data, 1 bit EOF. Parity bit is optional. . . . 73
4.14 Rising edge detection. The edge detection method can generate a high level signal lasting for 1 clock cycle after the red rising edge. It is used in combination with the clock rising edge highlighted in blue as the trigger signal for other operations. 74
4.15 State transition diagram of UART receiver R_1. 74
4.16 State transition diagram of UART transmitter T_1. 75
4.17 RTL model block diagram of UART-RS counter. Cs_1 to Cs_4 are clock synthesis module producing desired clock frequencies. 76
4.18 Behavioral simulation of UART-RS counter RTL model. 77
4.19 Communication experiment between host computer and FPGA hardened UART-RS counter. The last E1 is an end mark signifying the completeness of counting. 77

5.1 Target numerical P system. The nested (membrane) structure is represented by a Venn diagram; the variables and the rules are placed in corresponding locations; the initial value of variables follow them in square brackets. 82
5.2 Target enzymatic numerical P system. 83
5.3 The numerical P system generating Fibonacci sequence. 84
5.4 The projectile motion. The red solid line is the piecewise-linear approximation of the non-linear trajectory. 85
5.5 The NPS corresponding to the system of difference equations given in Equation 5.6. 85
5.6 The symbol object P system described in NPS form. A rewriting rule is translated in a set of programs in which programs involving subtraction denote obm_l and programs involving addition signify obm_r. Programs in parentheses are executed in membranes indicated by their subscripts, not in membranes containing them. 87
5.7 The effect of rules is equivalent to the set of programs in each membrane, where k_{i-j} denotes instance of rule r_{i-j}. Once k_{i-j} have computed, these programs are used to update multiplicity of objects. . . . 88

5.8 Linear approximation of $f(x) = x^2$ by $g(x) = |x|$. 90
5.9 The predator-prey GNPS. A predicates and its associated rule are placed in two rows in a brace to show them more explicitly. 90
5.10 Main procedures of FSPIP method. 92
5.11 GNPS system from Example 5.7. The predicate for each program is taken to a separate line before it. Variables in red and blue indicate input and output variables respectively. Others are intermediate variables. 93
5.12 GNPS model for case study 1. It implements the core computations of Sobel image edge detection algorithm. The predicate P_i and program Pr_i are written in two lines to render them better. GNPS1 has a skin membrane containing 4 programs and 19 real-value variables. Input variables are highlighted in red. Output variables are highlighted in blue. 94
5.13 Post implementation timing simulation of GNPS1. Port b_1, b_3 and b_4 obtain their steady output value after the eleventh rising edge of clock, indicating it costs 110 ns to get results. For b_2, its steady output value emerges after sixteenth rising edge, costing 160 ns to compute outcome. 96
5.14 Software simulation of GNPS1. It is assumed that GNPS1 evolves one step to stop. There is no one-to-one correspondence between a clock cycle and a GNPS step. For complex arithmetic computations, one step of GNPS requires more than one clock cycle. 96
5.15 Hardware debug of GNPS1. Input variables cannot be debugged so there is no clock signal. Values are represented in hexadecimal, 01f66 is 8038 in decimal. $8038 \div 2^{11} = 3.9248046875$, which is the value of b_4. 005eb is 1515 in decimal, $1515 \div 2^{11} = 0.73974609375$, which is the value of b_3. 97
5.16 GNPS model for case study 2 is numbered as GNPS2. The equations inside are the core computations of Sobel image edge detection algorithm. GNPS2 has 5 membranes and evolves 4 steps to reach halt condition. Programs in each membrane compute concurrently while each membrane execute serially. 98
5.17 Post implementation timing simulation of GNPS2. The real timing of b_2 and b_3 is a little different than expected. 98
5.18 Software simulation of GNPS2. It is assumed that GNPS2 evolves four steps to stop. CPU of the host computer costs 0.009306 s to get results. 99
5.19 Hardware debug of GNPS2. Values are represented in hexadecimal, 070cd is 28877 in decimal. $28877 \div 2^{11} = 14.10009765625$, which is the value of b_1. 17c5 is 6085 in decimal, $6085 \div 2^{11} = 2.97119140625$, which is the value of b_2. 99

6.1 This is a plan view of Pioneer 3 DX robot which is covered by a hinged deck on the top. The 16 rectangles in light blue are the sonar sensors surrounding the robot, just beneath the hinged deck. Sensors are arranged in two arrays in the front and in the rear. The layout of sensors in two arrays is identical. 103
6.2 The NPS controller performing control law of Formula 6.1, which will be implemented in FPGA. This numerical P system is called NPS1 below. 104

6.3 Expected timing waveform of NPS1 RTL model. Red lines indicate value transfers from $weightRight_i$ to $weightLeft_i$ and from *rw* to *lw*. Blue lines signifies parallel computing of programs in associated membranes. 106
6.4 RTL model of NPS1 consists of 7 modules, although NPS1 has 69 membranes. There is a one-to-one correspondence between membranes and programs inside. This correspondence transforms the implementation of a membrane to implementation a set of programs inside. Programs can be synchronized in one module with parallel constructs of Verilog. This is the reason why the number of modules can be reduced substantially. A *state* port is added to NPS1 so that it possesses idle and busy state. 108
6.5 Waveform of behavioral simulation of NPS1. The values of $weightLeft_i$ alternate as expected when *counter* loops its value. Sensors' readings s_i take the value in the last row of Table 6.1 which are abridged from waveform for the sake of taking a screenshot including computing results *rw-out* and *lw-out*. 109
6.6 Results of *PeP* simulation of NPS1. It costs 0.011703 seconds to obtain results. 109
6.7 Post implementation timing simulation of NPS1. 110
6.8 Hardware debug of NPS1. Numbers are represented in hexadecimal. . 110
6.9 Enzymatic numerical P system ENPS1. The value of enzyme e is larger than that of f in membrane 1, so these two programs can take place. Enzymes e_i have greater values than weight values so the 16 programs in membrane 2 can execute in parallel. 111
6.10 Waveform of behavioral simulation of ENPS1. Left and right wheel speed variables gain their expected values after the first rising edge. . 111
6.11 *PeP* simulation results of ENPS1. It costs 0.002993 seconds to obtain results. 111
6.12 Post implementation timing simulation waveform of ENPS1. 112 ns is regarded as the elapsed time to get results. 112
6.13 Hardware debug of ENPS1. 112
6.14 Obstacle avoidance kinetics analysis of Pioneer 3 DX robot. 113
6.15 GNPS3 which carrys out control law 2. 114
6.16 Post implementation timing simulation of GNPS3. The implemented clock period is 45 ns whose half period cannot defined so 44 ns and 46 ns are set as periods. So the elapsed time is the mean time which is 260.657 ns. 115
6.17 *PeP* simulation of GNPS3. It costs 0.020997 s to output results. 115
6.18 Hardware debug of GNPS3. Numbers are represented in hexadecimal. 115
6.19 RTL model of UART-NPS1. 117
6.20 Numbers are in hexadecimal for real numbers are represented in fixed point binary numbers. These data repeats 00 00 00, 26 D6 40 and 24 29 C0. Two digits of hexadecimal correspond to 8 bits in binary so three dual hexadecimal pairs comprise a 24-bit binary number. These three repeated data strings are 0, 310.6953125 and 289.3046875 in decimal, which are the desired three outcomes of *rw* of NPS1. 118

6.21 When input counter counts from 0 to 47, sensor readings are received into NPS1. Computing signal *cmp-state* converts from 0 to 1 to trigger computing. When left wheel speed are outputted, transmitting signal *tr-state* shifts from 0 to 1 to trigger transmitting. At the time when transmitting counter signal *tr-clken-cnt* has value of 5, *tr-state* switches its value to terminate transmitting. 118

7.1 Graphical representation of the target RRT. Black points indicates obstacle points and the red point denotes robot initial point. Blue points are random points and green points are generated RRT points. 125
7.2 The RRT-GNPS designed to execute RRT algorithm which generates two RRT points in eight obstacle points. It consists of 34 membranes including the skin membrane. 126
7.3 *PeP* simulation of RRT-GNPS which perform 34 steps and costs 0.097948 s to get results. The results of *PeP* save 2 significant digits. 127
7.4 The dynamic range is represented using radix 2 as well as radix 10. Zero is denoted by 1.0×2^0, $-\infty$ by -1.0×2^{128} and $+\infty$ by 1.0×2^{128}. . 127
7.5 Procedure diagram of the adder. This is a composite unit which can perform addition and subtraction. 130
7.6 Procedure diagram of the multiplier. 130
7.7 RTL model of the RRT-GNPS generating the first RRT point. 134
7.8 The whole process of RRT-GNPS RTL model characterized by UML activity diagram. 136
7.9 The behavioral simulation waveform of RRT-GNPS RTL model. The red waveforms are caused by the lack of initial values of variables. This phenomenon can be eliminated by assigning initial values to *reg* variables. 137
7.10 The physical constraints of RRT-GNPS shown in the package view of FPGA. Pins with orange bars inside signify the occupied pins. The light blue hexagon pins are clock pins while the gray round pins are ordinary input/output pins. 138
7.11 The stable value of first RRT point (x_2, y_2) appears at 1428979 ps (1428.979 ns) while the second RRT point (x_3, y_3)arises at 1630027 ps (1630.027 ns). So it costs $1428.979 + 1630.027 = 3059.01$ ns to get results. 138
7.12 Hardware debug of RRT-GNPS. Numbers are represented in hexadecimal. As can be validated, 411dc62f is the IEEE 754 format of decimal number 9.86088466644287. This result coincides with post implementation timing simulation result given in Figure 7.11. 139

A.1 Post implementation timing simulation of $\mathcal{R}_{ifl2}$. 145
A.2 Hardware resources dissipation and power consumption of $\mathcal{R}_{ifl2}$. . . . 146
A.3 Physical constraints settings of two heat shock response reaction systems. Gray circles with wide orange bars are pins allocated to RTL model ports. 146
A.4 Post implementation timing simulation of $\mathcal{R}_{hsr1}$. 147
A.5 Hardware resources dissipation and power consumption of $\mathcal{R}_{hsr1}$. . . . 147
A.6 Waveform comparison of interactive process 2 of $\mathcal{R}_{hsr} = (B_3, A_3)$. . . . 148

B.1 Block diagram of GNPS1 RTL model. After the design of a model, the corresponding schematic can be drawn automatically by Vivado. The schematic characterizes the same functions/behaviors as RTL model representing by HDL. 149
B.2 Total power consumption is the sum of device static power and dynamic power. Power consumption of the two cases are nearly the same, although GNPS1 works in all parallel and GNPS2 works in sequential. 150
B.3 Block diagram of GNPS2 RTL model. Each membrane is modeled in Verilog basic functional unit, *module*. The bug icons indicate variables to be debugged in *Hardware Debug* procedure. 150
B.4 I/O planning of NPS1 RTL model. NPS1 has one 1-bit input port *clock* and two 24-bit output ports *rw* and *rw*, so totally 49 pins are distributed to these three ports. 151
B.5 Hardware resource dissipation and power consumption of hardened NPS1. 151
B.6 I/O planning of ENPS1. 49 pins are used to represent I/O ports. . . . 151
B.7 Hardware resource dissipation and power consumption of hardened ENPS1. 152
B.8 I/O planning of GNPS3. 152
B.9 Hardware resource dissipation and power consumption of hardened GNPS3. 152
B.10 The resource utilization and power consumption of RRT-GNPS. 153
B.11 The *Place* & *Route* of RRT-GNPS. 153

List of Tables

3.1 Truth table of $y = (\bar{a} \wedge b) \vee \bar{c}$. 31
3.2 The comparison of the time-oriented and space-oriented conflict resolution 45
3.3 The differences of the conflict resolutions adopted in two design modes 47
3.4 The comparison of the rule-based and region-based design of Nguyen's implementation 47
3.5 The quantitative attributes of FPGA implementation 51
3.6 The qualitative attributes of FPGA implementations 52

4.1 The truth table of Formula 4.2. 57
4.2 The truth table for Example 4.7. 61
4.3 Molecular reactions and their RS equivalents of self-assembly intermediate filaments. 63
4.4 Interactive process of $\mathcal{R}_{ifl1} = (B_1, A_1)$ (6 steps) with T as the sole entity of each context for intermediate filaments RS. 63
4.5 RS reactions of $\mathcal{R}_{ifl2} = (B_2, A_2)$. 63
4.6 Interactive process (11 steps) of $\mathcal{R}_{ifl2} = (B_2, A_2)$. 66
4.7 Reactions of heat shock response RS $\mathcal{R}_{hsr} = (B_3, A_3)$. 69
4.8 Interactive process 1 (5 steps) of $\mathcal{R}_{hsr}$ at 37°C. 69
4.9 Interactive process 2 (5 steps) of $\mathcal{R}_{hsr}$ at 42°C. 69
4.10 RS reactions of $\mathcal{R}_{bc}$. 72

6.1 The calibrated values of $weightLeft_i$ and $weightRight_i$, along with a set of sampled sensors reading data which will be utilized to verify the correctness of RTL model of NPS1. 105

7.1 Exceptions and normalization of IEEE 754 single precision floating point number. 127
7.2 Determine the sign of the add/subtract result. 129
7.3 Comparison of two FP numbers with same signs. 131

Chapter 1

Introduction

Natural computing investigates natural processes in terms of computation and artificial computations inspired by natural processes [1, 2]. The topic of this thesis concerns hardware implementation of a qualitative model, reaction systems which concentrate on interactions of biochemical reaction processes taking place inside/outside biological cells and a quantitative computational model, numerical P systems inspired by the structure and function of biological cells, together with their real-life applications in different fields.

1.1 Biological cell: foundation of life

It is a fact that life is investigated without a definition which is acknowledged as a consensus. One clear point is that something constituted of biological cells (or one cell) is definitely a life. Creatures composed of cells are entitled as *cellular lives* [3, 4], such as a unicellular bacterium, a multicellular homo sapiens writing this thesis. Hundreds millions of years of submarine volcanism of primitive Earth provided continuous heat and special inorganic molecules in the primal ocean. These inorganic molecules reacted with others near the hydrothermal vent of volcanoes, producing some organic macromolecules, including proteins and nucleic acids. This assumption is partially validated by Miller-Urey experiment [5, 6]. At some moment some proteins and nucleic acids began to replicate themselves independently or collaboratively. These organic macromolecules are termed as *molecular lives* [7, 8] taking into account that they will evolve to cellular lives [9], or they parasitize in cellular lives. This is a hypothesis rather than a fact trying to understand the origin of life which convinces me.

Molecules replicate themselves under the control of physic laws other than subjective consciousness, although subjectivism is also the reflection of some physical/-mathematical theories [10]. Subject to limited resources, not all molecular lives can replicate and so exist for a long time, except they evolve some capabilities, structures or tools to win the replication war. It is assumed that all the creatures on earth originate from a molecular life named last universal common ancestor (LUCA) [11, 12, 13]. Deoxyribonucleic acid (DNA) and ribonucleic acid (RNA) composed of deoxynucleotides are extraordinary organic macromolecules which can capture dissociative deoxynucleotides because of hydrogen bonds in their basic groups [14, 15]. This assembly process produces another single-strand D/RNA chain. But this process can go wrong over time, causing mutations. Some mutant D/RNA macromolecules possess the ability to combine with amino acids to generate proteins. Some proteins comprise the protective shell or the instrument which plunders resources from other macromolecules. After a long time of evolution, an ultimate instrument comes into being: a cell. Since then, molecular life evolves to cellular life.

The membrane structure of a cell prevents other macromolecules entering to pillage deoxynucleotides and proteins. Requisite proteins can be produced on ribosomes in the cytoplasm. Endocytosis of membrane can also assimilate external proteins. For eukaryotic cells, DNA are protected by nuclear envelope aside from membrane. A cell is a powerful factory, a splendid device within which genes can be replicated safely and steadily. With the putative arising of protocells 3.84 billion years ago in hydrothermal vent precipitates, at which a time-line not long after the estimated forming of the Earth and oceans 4.54 and 4.41 billion years ago [16], cellular lives have had been existed and evolved until now. Nevertheless, it dose not mean that molecular lives are too fragile and eliminated by cellular lives. In effect, molecular lives are formidable and accompany with cellular lives for a quite long time. A good example of molecular life is a nightmare for human beings–virus.

A virus which is inanimate outside a cell is a RNA or DNA covered by protein capsid. Once it manages to enter a cell membrane, it suppresses gene expression of its parasitifer, taking advantage of organelles to produce its own genetic material and protein capsid, and assembling these stuff to produce new viruses [17]. These molecular lives are tenacious, but the basis of their existence and reproduction is cells, the same as cellular lives. By observing these special molecular lives, we may draw a conclusion that the evolutionary direction of molecular life is not necessarily the cellular life if there are other ways to replicate genetic materials. This conclusion helps us to think about what a life means. Considering the only commonality between a molecular life and a cellular life is reproduction, a matter which can reproduce itself may be regarded as a life. Cellular lives constitute a rooted tree (the root is LUCA), while viruses and other parasitical molecular lives comprise different 'rootless' parasitic vines twining on this tree.

1.2 Biological system modelling

Appreciating the function of an individual biochemical molecule is far from enough to elucidate the mechanism of the biological system in which the molecule works inside thoroughly. The primary reason is that molecules and their reactions are arranged in networks with various complexities [18]. As a result, the interactions of chemical entities and reactions play a more important role in the establishment of different functionalities and behaviors of a biological system. To gain a comprehensive understanding of inherently complex biological or ecological systems, conceiving and formulating mathematical models on the basis of large amount of experimental data is a practical approach.

Once a primitively mathematical model obtained, it can be simulated by programs in a computer (*in silico* simulation), which is typically a much more economical method to estimate biological systems than conducting biological experiments (*in vitro* simulation). By analyzing the discrepancies in the results between in silico and in vitro simulation, the mathematical model evolves gradually to a more refined one reflecting the biological system more accurate. In principle, there are two classes of biological systems/processes investigated: dynamic ones and quasi-stationary ones. The former one which is the majority is modelled by *ordinary differential equations* [19, 20] generally. Biological mathematical models modelled by this way are referred to as *quantitative* models because they concern about the concentrations of chemical entities over time precisely. On the contrary, there exist mathematical models which are not sensitive to the concrete concentrations of entities. In some special models,

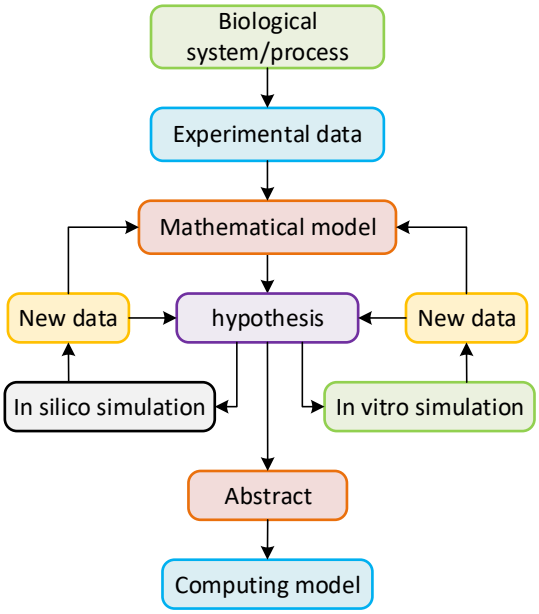


FIGURE 1.1: Biological system or process modelling and biological computing. The comprehensive investigation of biological systems or processes is enlightening for theoretical computer scientists as well, resulting in biological computing a cross-discipline filed.

what concerns is the causality effects between entities instead of their concentrations. These models are referred to as *qualitative* models.

From the perspective of computer science, some marvelous biological systems or processes are so powerful that computing models can be abstracted from them [21]. These abstractions are based on some hypotheses raised from mathematical model, accounting for the specific behaviors or processes of target systems. On one side, the simulation results are helpful to upgrade the mathematical models, so as to the hypotheses. On the other side, the improved hypotheses contribute to more sound computing model. The relationship between biological system modelling and biological computing is illustrated in Figure 1.1.

1.3 Reaction systems

Essentially, all the biological entities, e.g., the phospholipid molecules constituting plasma membranes [22], nucleotide molecules comprising DNAs [23], are products of a variety of biochemical reactions. The functioning of a biological cell is the manifestation of effects of considerable biochemical reactions taking place inside, together with effects of interactions among biochemical reactions. As a matter of fact, interaction of biochemical reactions is the dominant factor with respect to that reactants/products of some biochemical reactions facilitate/inhibit other reactions. Consequently, the interaction of reactions commences through facilitating/inhibiting mechanisms which regulate all biochemical reactions [24]. To investigate interactions of biochemical reactions, a formal framework entitled as *reaction systems* (RS) is proposed [25].

Contrary to most of the existing quantitative computational models such as Petri nets [26], reaction systems are a qualitative model working with sets instead of multisets of objects [27, 28]. There are two fundamental assumptions [29, 30] impart qualitative features to reaction systems: *threshold supply* assumption which means if

an element is present, then the amount of this element is adequate so that any number of reactions using this element as a reactant can carry out concurrently without conflict. Another important property is the *no permanency* assumption which means elements no longer produced by any reactions vanish except if they are imported from outside or their durations are larger than one [31, 32, 33, 34]. Furthermore, *threshold supply* assumption ensures no quantitative restrictions for consumption of reactants, hence reaction systems are deterministic and the elements competed by multiple reactions are distributed to all the reactions concurrently so that applicable reactions and their products are definite.

Due to the the fact that reaction systems are motivated by biochemical reactions and their interactions happening in cells, they are employed to model biological processes such as heat shock response [35], self-assembly of intermediate filaments [36], period-doubling bifurcation [37], etc. On the other hand, reaction systems can also be used to design quantitative models such as binary counters [29, 38] despite that they belong to qualitative framework. The trick of transforming qualitative model to quantitative lies in the perspective towards quantity of elements. Specifically, according to *threshold supply* assumption mentioned above, an element produced by some reactions has enough amount to suffuse a cell. Then this glutting state, rather than a single element, is regarded as binary 1. Otherwise it is deemed as binary 0. By this way, a reaction system can represent binary quantities [39, 40, 41].

1.4 Membrane computing

Eukaryotic cells have plasma membranes separating them from environment and diversified internal membranes delimiting organelles such as ribosome, mitochondrion and cytoblast [42]. The karyotheca of an eukaryotic cell prevents cell DNA from the infection by DNA viruses [14], while prokaryotic cells do not have nuclei so that bacteriophages can pierce membranes of bacteria to inject their DNA/RNA to infect them easily. From this fact we can see the importance of internal membranes. Receptor-mediated endocytosis [43], symport and antiport mechanisms impart communication capacity to membranes. A whole lot of biochemical reactions carry out in the cytoplasm as well as in all kinds of organelles simultaneously and interactively, producing indispensable entities required for cell proliferation and assembling new cells. From monoplast to multicellularity, the cellular lives remain unchanged since more than 3.8 billion years ago [16]. That is, the biological cell structures defined by membranes have evolved so being optimized for billions of years. As a consequence, a bio-cell is powerful parallel processing unit which can perform sophisticated biologic behaviors.

Enlightened by the insights of biological membranes and the biochemical reactions inside, and based on the foundation of molecular computing (especially DNA computing [44]), membrane computing was initiated by Gheorghe Păun in 1998 [45]. As mentioned in [46], membrane computing deepens molecular computing by introducing membrane structures, giving rise to the concept of "computing cell". Consequently, it is coherent to regard membrane computing models which are termed as P systems [47] as artificial cells being capable of computation. One basic aspect about membranes should be clarified, which is, unlike artificial neurons which are modeled as computing nodes in neural network, membrane computing does not model biological membranes as sorts of calculation devices. In effect, obtaining distributed space defined by membranes in which evolution rules in the form of artificial chemistry [48] can execute in parallel to consume and produce objects is the main goal

of the existence of membranes. These artificial membranes have neither material concentration nor internal structures analogous to biological ones, just a abstract concept.

Membrane computing models, i.e. P systems are a class of quantitative models working on multiset of objects from an alphabet. This quantitativeness originates from the competitive distribution of objects (analogous to molecules in bio-cells) among multiple applicable rules (if there are) in a region delimited by related membranes. In a nutshell, resources (objects) are limited, they should be allocated to all applicable rules to consume resources as much as possible. The contest of objects brings non-determinism into P systems because generally there are several candidate applicable rules to be used in a membrane. It is suitable to assign equiprobability to these rules to be chosen. Large scale parallelism and non-determinism are two properties which fling down challenges for simulation/implementation of P systems. The theme of this thesis is trying to cope with this challenge to some extend.

Inspired by the structure of cells, tissues composed of cells and neurons, there are three classes of P systems corresponding to these cellular structures: cell-like P systems [49], tissue/population-like P systems [50, 51, 52] and spiking neural P systems [53, 54, 55]. Due to its background inspired from, membrane computing has worked as a modeling framework for biological and ecological subjects such as modeling artificial life [56], photosynthesis [57, 58], protein signaling pathway [59], cell-mediated immunity [60], Catalan Pyrenees bearded vulture ecosystem [61] etc. Please refer to [62] for more membrane computing applications in computer science and linguistics. On the other hand, the inherent large scale parallelism of membrane computing has the profound potential for the progress of extreme data processing. This thesis focuses more on engineering applications of membrane computing, especially robot control and path planning based on a special type of cell-like P systems called as numerical P system (NPS) and its variants.

1.5 FPGA based reconfigurable computing

Reconfigurable computing [63] concerns about the reconfigurability of hardware circuit datapaths besides control flows [64, 65]. The high flexibility compared to application specific integrated circuits (ASICs) and high performance (along with low-power consumption) with respect to software programmed general-purpose processors (i.e., CPUs) provided by reconfigurable architectures are the principal superiority of reconfigurable computing. Roughly speaking, there are three levels of hardware arrangements in reconfigurable computing realm, which from lowest to highest are: reconfigurable computing device level, architecture level and system level [66]. The celebrity of reconfigurable computing devices is the field programmable gate arrays (FPGAs). Their popularity stems from the fact that they are predominantly off-the-shelf reconfigurable computing devices, if they are not the only alternative.

Reconfigurable architectures can be constructed with custom-designed chips different from FPGAs. The Garp reconfigurable processor [67, 68] is a fine-grained architecture allowing bit manipulation resembling FPGAs. PipeRench [69, 70] is an arithmetic logic unit (ALU) based coarse-grained architecture working as a coprocessor. Reconfigurable processing fabrics (RPFs) are the crucial constructs in these

reconfigurable architectures. RPFs can be integrated into traditional microprocessor-based systems as separate coprocessors or functional units coupled with CPU. Independent RPFs are devised and implemented as coprocessors in [71, 72, 73]. RPFs can be loosely coupled as such the architecture in [74] or tightly coupled like that in [75, 76, 77].

Reconfigurable computing systems are built from reconfigurable devices and architectures. The Fixed-Plus-Variable (F+V) structure computer [78, 79, 80] is known as the first reconfigurable computing system (machine). Several FPGA based reconfigurable systems were developed after FPGAs are easily available in 1980s, such as Programmable Active Memories (PAM) project [81], Virtual Computer [82], Splash system [83], PRISM [84], Configurable Array Logic (CAL) [85], etc.

Nowadays, FPGAs are highly isomerized with RAMs, fast carry chain, multipliers, DSPs and even CPU blocks, resulting a system-on-a-chip, rather than simple devices as in their early stage. So a current high-end FPGA is a reconfigurable system. As can be seen in the later chapters, the cell-related computing models are parallel heterogeneous for different syntactic components computing simultaneously. However, the inherent parallelism of such models is ill suited to be implemented on the general purpose CPUs since we are unable to confirm that whether the computing processes performed by the CPU complies with theoretical models or not. Nor do we know that the parallelism of a CPU brought about by the multi-core architecture is adequate to execute these models. Implementing these newly cell-inspired heterogeneous application-oriented models on FPGAs or other reconfigurable architectures/systems which are also parallel isomerized is more preferable for their high degree of parallelism and reconfigurability.

Because of the availability of commercial FPGAs, two FPGA developing boards, Digilent BASYS 3 and Xilinx VC707, featured with a small capacity FPGA and a modest capacity FPGA respectively are selected as target hardware to implement several reaction systems and NPS models. The control flow and data flow modeled on target FPGAs are almost identical to theoretical models so that the parallelism is exploited as much as possible. Overall, resorting to reconfigurable computing to implement cell-related and application-oriented computing models is the substance of this thesis.

1.6 Thesis statement

1.6.1 Main research contents

This thesis mainly studies the hardware implementation of two parallel computing models inspired by the structures, behaviors and internal biochemical reactions of biological cells, as well as the practical application of the implementation methods in different fields. Specifically, the FPGA implementation methods of reaction systems and numerical P systems (NPS) are proposed and applied in biological process modeling, robot control and path planning domains. When a parallel biological computing model is simulated on a general-purpose computer, it is not guaranteed that CPU performs the operations just in accordance with the theoretical model since the processing of CPU is not transparent. In fact, it is likely that the limited parallelism of CPU does harm for the intrinsic parallelism of biological computing model to a large extent. To fully realize the parallel computing process of theoretical models, the best choice is to run this kind of models on parallel hardware platform.

With the reconfigurability and parallelism of FPGA, reaction systems and numerical P systems concerned are transformed into the digital circuits in FPGA. When

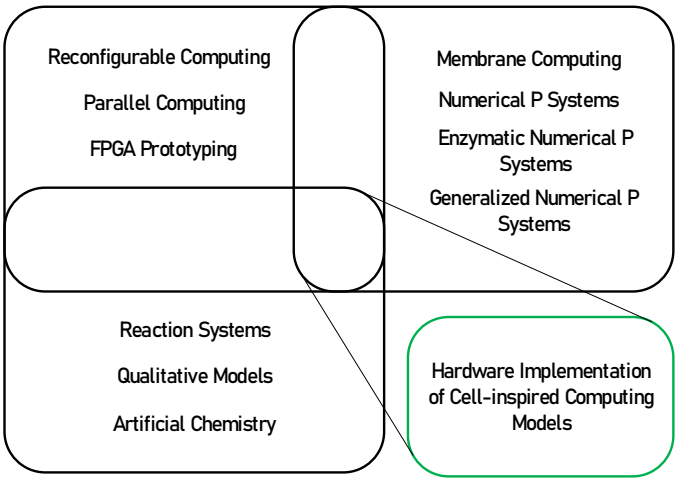


FIGURE 1.2: Realms involved in the investigations of this thesis. Reaction systems and P systems are tightly relevant while several models of them will be implemented on FPGAs as reconfigurable computing instances.

target FPGA has sufficient hardware resources, the calculation can be carried out in strict accordance with the procedures of theoretical models and accelerate calculations substantially. According to the proposed FPGA implementation methods of these two types of models, several models with great application value have been realized. This investigation can serve as a good reference for the exploration of newly parallel hardware architectures. Topics covered in this thesis include reaction systems, membrane computing (NPS realms), reconfigurable computing, and the cross-disciplinary area of the mentioned topics, as shown in Figure 1.2. The research contents include:

- In order to grasp the research status of the hardware implementation of reaction systems and P systems comprehensively and deeply, a detailed scientific research investigation was carried out. Consequently, the literature review of CPU simulation and CUDA-GPU implementation of reaction systems, and hardware implementation of symbolic object (cell type) P systems were obtained. Hardware implementation of reaction systems has not been reported in the literature as yet. Researches on the hardware implementation of P systems are analyzed particularly, which points out the direction for FPGA implementation method of numerical P systems.

- On the theoretical facet, for the sake of better comparing different methods and viewing these methods from a unified perspective, the object distribution problem of symbolic object P systems is described as a multi-criterion optimization problem. The relationship between symbol object P systems and NPSs is pointed out, by translating string rewriting rules as programs renewing cardinality of objects in parallel. The relationship between reaction systems and synchronous circuits is analyzed. Under certain conditions, reaction systems and synchronous circuits can be converted to each other. The relationship between the numerical P systems and systems of difference equations is investigated. It is found that numerical P systems are another manifestation of systems of difference equations, and the two can also be transformed into each other under certain conditions. Considering the relationship between systems

of difference equations and systems of differential equations, NPSs can model many engineering application problems.

- At the method aspect, based on the relationship between reaction systems and synchronous circuits, the FPGA implementation method of the reaction systems is proposed. In practical applications, FPGA hardened NPSs are used as control/process units involving input and output of sensors' data and computing results and program usability criteria in enzymatic numerical P systems (ENPS) are quite limited. To pursue an efficient implementation method of (E)NPS, extend NPS to generalized numerical P system (GNPS) whose variables are classified as input variables, output variables and internal variables. Presburger arithmetic is introduced in GNPS as well to extend program usability criteria. Further more, FPGA implementation method of GNPS is put forward. GNPS is a superset of NPS and ENPS, so NPS and ENPS can also be implemented in FPGA with the method devised for GNPS.

- According to the reaction system FPGA implementation method, reaction system models for the self-assembly of eukaryon's intermediate filaments, heat shock response reaction systems and reaction system modelled binary counter were implemented in FPGA. The computing speed of the FPGA hardened reaction systems is up to 2×10^8 steps per second.

- To verify the feasibility and correctness of the GNPS FPGA implementation method, a GNPS was designed to execute the core formulas of Sobel image edge detection algorithm and this GNPS was implement in FPGA. Contrasting with the results calculated by NPS simulation software, the GNPS FPGA implementation method is validated. Then the wheeled robot membrane controllers based on NPS, ENPS and GNPS were implemented in FPGA in line with this method. Compared with the CPU simulation execution speed, the speedup obtained with FPGA implementation has the order of 10^4 magnitude.

- Rapid-exploring random tree (RRT) is a computation-intensively parallel path planning algorithm. In order to accelerate the calculation of this algorithm, it is modeled in a GNPS which organizes all the parallel steps in accordance with the algorithm. Then this RRT-GNPS was implemented in FPGA. Taking into account the potential application of RRT algorithm in large-scale maps, the IEEE 754 standard of floating point (FP) numbers is adopted in the FPGA implementation. Arithmetic calculations units such as adder, subtractor, multiplier and reciprocal square root calculator are designed to obey this standard. For divisions which cannot be avoided, the FP division IP cores are instantiated to do divisions. Compared with CPU simulation, the speedup obtained by FPGA implementation also reaches the order of 10^4.

Main research topics and their relations are illustrated in Figure 1.3. In this figure, blocks at the same level connected with solid lines are equivalent models, while those connected with dashed lines can be approximated with some mathematical methods. Blocks in the upper level are generalized models of blocks in the lower level.

1.6.2 Contributions

The major contributions of this work consists in the following aspects:

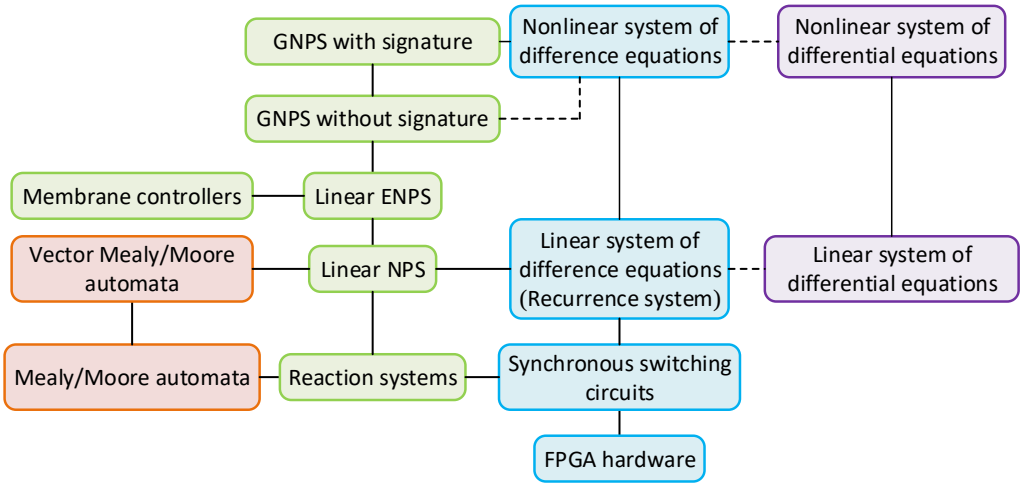


FIGURE 1.3: Relations of main research topics.

- An exhaustive analysis of all existing approaches of hardware implementation in the area of P systems is performed. The quantitative and qualitative attributes of FPGA based implementations and CUDA-enabled GPU based simulations are compared to evaluate the two methodologies. To better compare the different used techniques, the core of the simulation process is described as a multi-criteria optimization problem, that allowed us to have a uniform view on different approaches.

- A tight relation between reaction systems and synchronous digital circuits is shown. The attributes of reaction systems as a sort of qualitative model are studied. Based on the relation and attributes, the FPGA implementation method of reaction systems is proposed. By this method, simulations of large scale reaction systems can be accelerated to a large extent comparing with software simulation.

- The relation of NPS and difference/recurrence systems is analyzed, extending the original definition of NPS to generalized NPS (GNPS) to facilitate hardware implementation. The differences between a symbol object P system and a NPS is analyzed and shown explicitly. FPGA implementation method of GNPS is conceived, which can speedup computations in robot control and path planning to an order of 10^4 to 10^5 magnitude.

- The FPGA implementation method of GNPS for IEEE 754 floating point (FP) numbers is proposed. FP calculation cannot be carried out using HDL arithmetic operators, but needs instantiating FP arithmetic modules to execute. However, these instantiated modules cannot be placed in HDL conditional statements, so the availability of GNPS rules cannot be determined by calculating such logical expressions. The module sequential triggering method based on edge detection is designed to handle this hurdle, which completes computing procedures in line with the sequence determined by conditional statements of GNPS. By adding appropriate delays, multiple FP modules perform their calculations correctly in the order of arithmetic priority. Parallel processing is realized by triggering calculations of all available rules through the rising edge of clock.

1.7 Thesis organization

This thesis is arranged as follows:

Chapter 1 introduces biological notions, biological system modelling, reaction systems, membrane computing and P systems, and reconfigurable computing based on FPGA, which are the research background of this thesis.

Chapter 2 presents basic definitions of reactions systems, P systems and also of numerical P systems and enzymatic numerical P systems. Then present how to convert the rule choice problem in P systems to an multi criterion optimization problem. The relations between symbol-object P system and NPS are studied as well.

In Chapter 3, CUDA-GPU and FPGA are introduced at first for they are the two devices involved in software simulation and hardware implementation of reaction systems and P systems. FPGA development flow of Xilinx integrated development environment, Vivado, is also given. The CPU simulation and CUDA-GPU implementation of reaction systems, and FPGA implementation of P systems are reviewed. Taking stock of the complexity of FPGA implementation of P systems, the research background of membrane computing hardware implementation is introduced firstly. Then the general ideas of its hardware implementation are explained, shedding light on the representation of P system in FPGA, the indirect method and the direct method for object distribution problem, and algorithms for addressing non-determinism briefly. From two aspects of region based and rule based implementation strategies, the existing FPGA implementation researches of symbol object (cell-like) P systems are analyzed in detail, and different FPGA implementation strategies of P systems are estimated from the quantitative and qualitative perspective. Finally, the microprocessor or controller based implementations of P systems are surveyed.

In Chapter 4, the relationship between reaction systems and synchronous circuits is investigated. FPGA representation and implementation of reaction systems are put forward. Three reaction systems having distinct functions are implemented in FPGA, achieving a considerably fast speed.

Chapter 5 analyzes the relationship between NPS and systems of difference equations, defining the definition of binary and unary normal form for (E)NPS. Next the notion of GNPS is introduced. FPGA implementation method of GNPS is devised and two case studies are implemented to confirm this method.

In Chapter 6, according to the FPGA implementation method of GNPS, three wheeled robot membrane controllers based on NPS, ENPS and GNPS are implemented. NPS membrane controller contains variable value transmissions. Counters were used to count the calculation step, so specific values were transferred at correct step. In the implementation of ENPS membrane controller, the logical expressions were used to express the conditional statements involving enzymatic variables. The control law of GNPS controller is relatively complex, but the computation process completes speedy for the parallelism of GNPS. NPS and its variants covered in this chapter deal with fixed-point numbers, so HDL (Verilog) arithmetic operators are utilized.

Chapter 7 studies FPGA implementation of GNPS based RRT algorithm. A GNPS performing all the parallel computations is designed and implemented according to the method proposed in Chapter 5. Considering the potential large-scale application, GNPS deals with floating point numbers conforming to the IEEE 754 standard. In order to avoid the division operation with the flaw of high resource consumption and slow computing speed, the reciprocal square root unit is designed to calculate the quadratic formulas under the fractional line. For inescapable divisions,

the floating-point division IP core optimized for performance was instantiated. The execution of rules is accomplished by instantiating multiple arithmetic units. As instantiated modules, they cannot respond to the results of logical expressions representing the criteria for rule availability. According to the calculation sequence of arithmetic operations, the edge detection trigger method is designed to realize the function of conditional statements so that calculations are performed rightly.

Conclusions are summarized in Chapter 8.

Chapter 2

Definitions

Both reaction systems and P systems are living cell inspired models. Reaction systems are qualitative models focusing on biochemical reactions, especially their interactions. While P systems are quantitative models concentrating on distributed compartments delimited by membranes in which artificial chemical reactions can be carried out concurrently and non-deterministically. This chapter elucidates general definitions of these two models and concerned variants of P systems.

2.1 Reaction systems

Biochemical reactions take place in cells of cellular lives all the time, comprising the complicate functionality of cells. Parallel executing biochemical reactions interact with biochemical entities (reactants) in the environment in the form that some of these entities prevent some reactions from happening or vice versa, catalyzing some of the reactions. The inhibition and facilitation mechanisms are the main regulation principles for reactions in living things. Reaction systems, initiated in 2004 [24], are a formal framework investigating interactive processes which come into being on the foundation of inhibition and facilitation mechanism of biochemical reactions.

There are two assumptions underlying the qualitative trait of reaction systems:

- *Threshold supply* assumption: if a kind of element is present, its quantity is sufficient so that multiple reactions utilizing it as one of the reactants can take place simultaneously, without contending of this kind of element. As a result, there are no conflicts among reactions if their reactants are present.
- *No permanency* assumption: if a kind of element is not taken in (from environment or other reaction systems), it persists if there is some reactions producing it. The default duration of a element [29] is "one period"–the time interval from one state to the next. This sort of element vanishes beyond this time if it is not a product of any reactions. After the execution of applicable reactions, all the reactants are transformed to corresponding products, resulting reactants sets vanishing to empty sets.

A reaction system is deterministic for all reactions involving the same reactants can proceed reacting due to *threshold supply* assumption. Then all the reactions that should happen will happen, there is no need to make choices among applicable reactions. Consequently, the behavior and the products of a reaction are definite for reaction sets in each state are determined according to reactions and initial conditions of elements. Now the formal definition of these notions is given below.

Definition 2.1. [25] *A reaction is a triplet $r = (R_r, I_r, P_r)$, where R_r is the reactant set, I_r is the inhibitor set, P_r is the product set. If there is a set B such that $R_r, I_r, P_r \subseteq B$, then r is a reaction in B. $rac(B)$ signifies all reactions in B.*

I_r can be empty while R_r and P_r should not be empty in general. It is pointed out that a special reaction $\Phi = (\emptyset, \emptyset, \emptyset)$ is termed as *empty reaction* may be used.

Definition 2.2. [35] *Let T be a finite set and A be a set of reactions,*

1. *the* result *of applying r on T, denoted by $res_r(T)$, is defined as*

$$res_r(T) = \begin{cases} P_r & \text{if } R_r \subseteq T \text{and } I_r \cap T = \emptyset, \\ \emptyset & \text{otherwise.} \end{cases}$$

2. *the* result *of applying A on T, denoted by $res_A(T)$, is defined as*

$$res_A(T) = \bigcup_{r \in A} res_r(T)$$

Definition 2.3. *A reaction r is applicable on a set T if $res_r(T) \neq \emptyset$, i.e., $R_r \subseteq T$ and $I_r \cap T = \emptyset$, or else r is inapplicable on T. A reaction set A is applicable on T if each reaction in A is applicable on T.*

Now the static structure of a reaction system can be defined below. The definition of the reaction is given preceding the definition of the reaction system on account of that a reaction system is essentially a set of reactions.

Definition 2.4. [25] *A reaction system is an ordered pair $\mathcal{R} = (B, A)$, where $A \subseteq rac(B)$ and B is a finite set termed as the* background set *of $\mathcal{R}$.*

Definition 2.5. [25] *For a reaction system $\mathcal{R} = (B, A)$ and a set $T \subseteq B$, define $res_{\mathcal{R}}(T) = res_A(T)$ and $en_{\mathcal{R}}(T) = \{r \in A \mid r \text{ is applicable on} T\}$.*

The dynamic process of a reaction system is characterized by *interactive process* which is defined as follow.

Definition 2.6. [25] *If $\mathcal{R}$ is a reaction system, an interactive process in $\mathcal{R}$ is a pair $\pi = (\gamma, \delta)$, where $\gamma = C_0, C_1, \ldots, C_n$, $\delta = D_0, D_1, \ldots, D_n$, $n \geq 1$, $D_0 = \emptyset$, $D_i = res_{\mathcal{R}}(C_{i-1} \cup D_{i-1})$, $1 \leq i \leq n$.*

Sequence γ is called *context sequence*, denoted by *con(π)*. Sequence δ is named *result sequence*, denoted by *res(π)*. Let $W_i = (C_i \cup D_i)$, $0 \leq i \leq n$, $n \geq 1$. $W_0 = C_0$ for $D_0 = \emptyset$. Sequence $\tau = W_0, W_1, \ldots, W_n$ is entitled as *state sequence*. Let $E_i = en_{\mathcal{R}}(W_i)$, sequence $\sigma = E_0, E_1, \ldots, E_n$ is called *activity sequence* of π.

Context sequence γ can be deemed as elements coming from environment or from other reaction systems, signifying the interactiveness between a reaction system and its surroundings. The transition process of states, W_i, can be counted as computation process of a reaction system, denoting by state sequence τ. The power source of transitions is activity sequence σ while results of transitions are exhibited by result sequence δ. The existence of these results obeys *no permanency* assumption. As a type of qualitative models, reaction systems work on sets instead of multisets of elements.

2.2 P systems

2.2.1 Symbol object cell-like P system

As mentioned in Chapter 1 Section 4, there are cell-like P systems, tissue/population-like P systems and spiking neural P systems. Cell-like P systems are the subject of

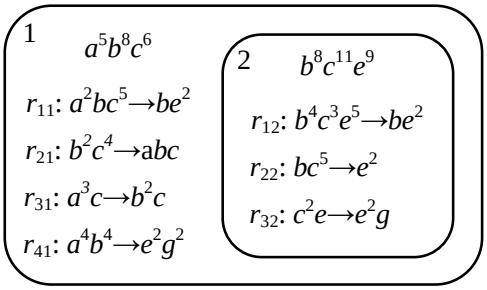


FIGURE 2.1: An example of a P system with two membranes. Membrane 1 contains a object multiset $a^5b^8c^6$ and four rules. Membrane 2 encompasses a object multiset $b^8c^11e^9$ and three rules.

this thesis. A symbol-object cell-like P system is an individual artificial cell with multisets of objects coming from an alphabet, such as Roman or Greek alphabet, and rewriting rules transforming multisets to other ones. Symbol-object cell-like P system is the classic model first introduced [45]. So when only "P system" is mentioned, it is implied that a symbol-object cell-like P system is under consideration. The static structure of a (symbol-object cell-like) P system is defined as:

Definition 2.7. [46] *A P system is a construct*

$$\Pi = (A, \mu, \omega_1, \ldots, \omega_n, R_1, \ldots, R_n)$$

where

1. *A is an alphabet whose symbols are used to indicate objects.*
2. *μ is the membrane structure consisting of n membranes;*
3. *ω_i, $1 \leq i \leq n$, $n \geq 1$, is the multiset of object located in membrane i;*
4. *R_i, $1 \leq i \leq n$, is the rewriting rule set associated with membrane i which transforms its object multiset.*

A membrane structure μ is a rooted tree (an undirected graph which is acyclic and connected), with the root node called *skin* membrane. Nodes in this structure distinct from the root node are (inner) membranes. Two nodes connected by one edge have a containment relation that the membrane denoted by the upper node include the membrane denoted by the lower node. The Venn diagram is used to represent the membrane structure so the containment is explicitly given by inclusion of membranes. The nodes that do not connect to any other nodes are the *leaves*, which are called elementary membranes containing the computing result typically. A label is attached to every membrane, so the labels can be regarded as nodes. Figure 2.1 depicts an example of a P system.

The inclusion of each membrane consists of multisets of symbol objects and reaction rules, which are named as (*evolution*) rules conventionally, which are called *rules* for short in the sequel. Each multiset of objects indicates the symbols and their numbers (multiplicities) in the multiset, such as $a^3b^7c^4$, where a, b, c are objects and their exponents designate their multiplicities, i.e., there are 3 as, 7 bs and 4 cs in this object multiset. Note that what provided here is just an example, there is no limit for the types and quantities of objects in a object multiset. The family of multisets over alphabet A is denoted as $M(A)$. On the basis of the membrane structure notion

stated above, a *cell* over an alphabet is a pair (μ, Map), where μ is a membrane structure, Map is an *mapping* from the set of nodes (represent membranes) $K(\mu)$ to $M(A)$. The *mapping* is the distribution of symbol object multisets to membrane structures.

The value of ω_i at a time instant comprises the *configuration* of a P system at that instant. For the P system in Figure 2.1, the current configuration is $\{a^5b^8c^6, b^8c^11e^9\}$. Rules are of the form $obm_l \rightarrow obm_r$ (e.g., $a^2b^3c \rightarrow e^2f^2$) where obm_l denotes reactant object multiset and obm_r indicates product object multiset. Bear in mind the form of reactions in reaction systems, we can see that rules of P systems are artificial reactions without inhibitor sets. P systems with promoters/inhibitors are investigated in [86, 87, 88]. Writing rules in the form of reactions in RS, a rule l is a 2-tuple $l = (R_l, P_l)$ where R_l denotes obm_l and P_l signifies obm_r. There is a one-to-one correspondence between a membrane and its internal space called *region*. So these two terms are treated equally without discrimination. It is emphasized here that reaction systems use sets of object for rewriting (the mapping is from a set), while P systems utilize multisets of object. A set can only describe qualitative information, while a multiset is suitable for quantitative relationship. The definition of applicability of rules is given below.

Definition 2.8. *A rule in a membrane is applicable if $obm_l \subset OM$, where OM is the object multiset in the same membrane. Otherwise this rule is inapplicable.*

Yet as a type of quantitative models, the applicability of a rule is determined in accordance with the quantitative relation between object multiset ω_i and each obm_l of every rule in R_i: if the quantity of each type of object in obm_l of a rule is no larger than that of the ω_i, then this rule is applicable; otherwise this rule is inapplicable. The quantitativeness of P systems asks for one thing more than the determination of applicability of rules, that is, the number of times termed as *instance* [89] an applicable rule can be used. A membrane may contain multiple applicable rules at the same time, leading to a set of applicable rule multisets denoted by $All(\Pi, C, \delta)$ which will be elucidated next. A terminology *derivation mode* refers to how all the applicable rules should be used. For example, if consuming ω_i as many as possible in a maximally parallel way so that the remainder object multiset is not adequate for any rules is the goal, then this derivation mode is named *max*. The *max* mode can be defined as follows [90]. A more formal definition of *max* mode will be given based on *Pareto dominance* later.

$$Applicable(\Pi, C, max) = \{R \subseteq Applicable(\Pi, C, max) \mid \nexists R' \in Applicable(\Pi, C, max) : R' \supsetneq R\}.$$

For a multiset of applicable rules derived by *max*, diminish each instance of rules to one obtaining a new multiset of applicable rules. If such multisets of applicable rules are applied, we say this P system evolves in *set-max* mode.

The *max* mode brings about a interesting but nerve-wracking situation—nondeterminism. To be specific, applicable rules confirmed according to *max* in a membrane are dependent, augmenting the instance of one rule causes the instance descending of another. The combinations of applicable rules bring about multiple solutions, viz., multiple applicable rule multisets ($All(\Pi, C, \delta)$), to evolve configuration of P system. It is equiprobable to choose one rule multiset to apply but which one will be chosen is stochastic. The computation process of P systems is signified by the transition of configurations. A P system reaches halting configuration if no rules can be applied in that configuration.

To expound a part of the computation process of a P system, take a concrete example depicted in Figure 2.2. Assume that this P system works *max* mode so one

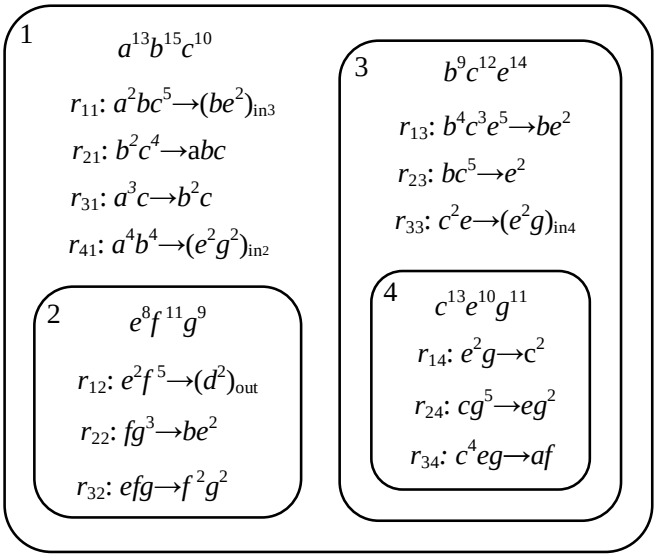


FIGURE 2.2: A P system at its initial configuration. The subscript in_3 of obm_r in rule r_{11} means obm_r will be sent to membrane 3, so do rule r_{14}, r_{33}. While subscript out of obm_r in rule r_{21} signifies its obm_r will be sent out to environment. This is a static P system that there is no evolution rule which can dissolving or creating membranes.

applicable rule can be used any possibly times, resulting a set of rule multisets. Applicable rule multiset in each region had been computed as illustrated in Figure 2.3. The configuration after the application of selected rule multiset is given in Figure 2.4. There are several solutions in each region for calculating the rule multiset is a multiple-solution problem. The non-determinism of P systems reflects this fact, besides the parallelism, since all the solutions are feasible solutions in accordance with the derivation mode. Only one rule multiset in each region is chosen equiprobably to comprise the solution denoted by $Applicable(\Pi, C, max)$ to evolve the configuration.

Suppose that the rule multisets in red font in Figure 2.3 are selected while rule multisets in black are unselected feasible solutions. After applying $Applicable(\Pi, C, max)$, the configuration evolves taking the form of consumption/production of object multisets. On the circumstance of the dissolving of membranes happening, the rules inside will disappear, otherwise rules keep unchanged during the transitions. The remained objects are in blue font and rules constituting rule multisets and their instances used (denoted by exponents above rules) are in red as shown in Figure 2.4. Notably, a rule multiset dose not necessarily contains every rule in a membrane. It can be verified that there are no applicable rules in the current configuration anymore, so the P system reaches its halt configuration which is denoted by $Halt(\Pi, C, \max)$. In practical applications, one can stop the evolve process when the results needed are generated even though the halt configuration is no reached.

2.2.2 Network of cells

The definition of a P system can be abstracted a step further to the notion of *network of cells* [91, 92, 93, 94]. Most types of (static structure) P systems can be seen as variants of parallel multiset rewriting (by using the algorithm called *flattening*). Using this underlying idea, any concrete variant of a P system could be described by specifying the following notions (functions):

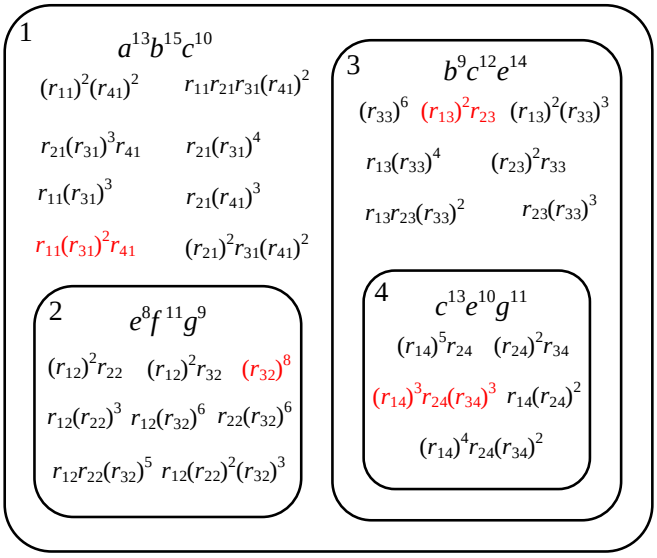


FIGURE 2.3: All the feasible rule multisets of each membrane. The selected applicable rule multisets are highlighted in red.

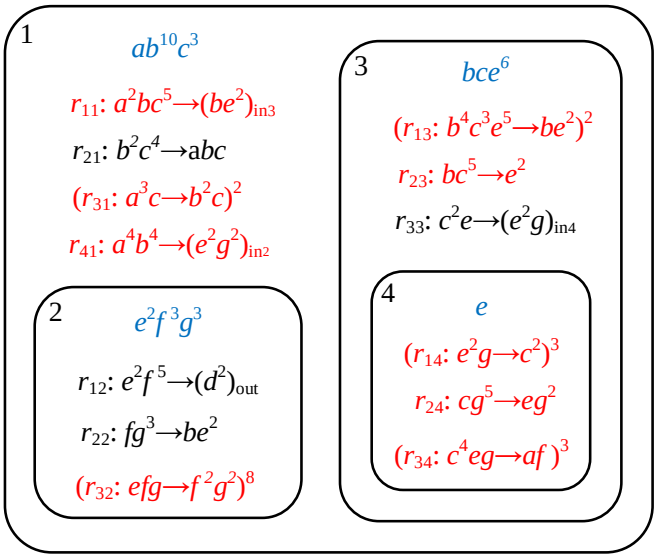


FIGURE 2.4: Rules in red are those constituting the applied rule multiset. The exponents above rules denote thier instances in the rules multiset. Residuary object multisets after the application of rule multisets are shown in blue.

- $Applicable(\Pi, C, \delta)$ – the function taking a system Π, a configuration C and a derivation mode δ to yield the set of multisets of rules of Π that can be applied to C.
- $Apply(\Pi, C, R)$ – the function allowing to compute the configuration obtained by the parallel application of the multiset of rules R to the configuration C.
- $Halt(\Pi, C, \delta)$ – a predicate that yields true if C is a halting configuration of the system Π (in some derivation mode δ).
- $Result(\Pi, C)$ – a function giving the result of the computation of the P system Π when the halting configuration C has been reached.

Since multiset rewriting level is not practical for system description and understanding, a higher-level concept called *network of cells* was introduced in [91]. This model augments multiset rewriting with the notion of spatial locations (*cells*) as well as the corresponding operations and it can be seen as a particular interpretation of the symbols. The works [91, 92, 93] define some basic building blocks in terms of network of cells and give several examples of the construction of widespread notions and types of rules in membrane computing using these blocks. Hence, a general framework, called the *formal framework of P systems*, is obtained allowing to express notions related to P systems in a common formal frame. Moreover, in most of the cases the obtained model (in the formal framework) allows a strong bi-simulation with the original one, thus they become indistinguishable. This permits several applications. A further evolution of the framework [95] permitted to take into account notions related to P systems with dynamically evolving structure.

The definition of *network of cells* provided below is taken from [91]. It is remarked that the definition from [95] is slightly different, however both models coincide when the structure of the system does not evolve.

Definition 2.9. [91] *A network of cells of degree $n \geq 1$ is a construct*

$$\Pi = (n, V, w, Inf, R)$$

where

1. *n is the number of cells;*
2. *V is an* alphabet*;*
3. *$w = (w_1, \ldots, w_n)$ where $w_i \in V^\circ$, for all $1 \leq i \leq n$, is the* finite multiset initially associated to cell *i;*
4. *$Inf = (Inf_1, \ldots, Inf_n)$ where $Inf_i \subseteq V$, for all $1 \leq i \leq n$, is the* set of symbols occurring infinitely often in cell *i (in most of the cases, only one cell, called the* environment*, will contain symbols occurring with infinite multiplicity);*
5. *R is a finite set of rules of the form*

$$(X \to Y; P, Q)$$

where $X = (x_1, \ldots, x_n)$, $Y = (y_1, \ldots, y_n)$, $x_i, y_i \in V^\circ$, $1 \leq i \leq n$, are vectors of multisets over V and $P = (p_1, \ldots, p_n)$, $Q = (q_1, \ldots, q_n)$, p_i, q_i, $1 \leq i \leq n$ are finite sets of multisets over V. The following form is also used (omitting p_i, q_i, x_i or y_i if they are empty)

$$(1, x_1) \ldots (n, x_n) \to (1, y_1) \ldots (n, y_n)\,; [(1, p_1) \ldots (1, p_n)]; [(1, q_1) \ldots (n, q_n)].$$

The above rule is applied as follows: objects x_i from cells i are rewritten into objects y_j produced in cells j, $1 \leq i, j \leq n$, if every cell k, $1 \leq k \leq n$, contains all multisets from p_k and does not contain any multiset from q_k.

The *configuration* C of Π is defined as an n-tuple of multisets over V $(u_1, \ldots, u_n)$ satisfying $u_i \cap Inf_i = \emptyset$, $1 \leq i \leq n$. The *computation* in networks of cells is a finite sequence of transitions, corresponding to the (parallel) application of one or several rules, starting in an initial configuration and ending in a final configuration (for which the *Halt* predicate returns true).

At each step, the set of multisets of applicable rules, denoted as $Applicable(\Pi, C, \delta)$, is restricted by the derivation mode denoted by δ, which specifies which sub-multisets are chosen to transit configuration.

A clear-cut definition of maximal parallel derivation mode (*max*) in a P system can be presented based on the notion of *Pareto dominance relation* [96]. The concept *dominate* among object multisets should be given firstly.

Definition 2.10. *An object multiset O_1 dominates object multiset O_2 if*

1. *the multiplicity of each object (symbol) in O_1 is no less than that of object in O_2;*
2. *at least the multiplicity of one type of object in O_1 is larger than that of object in O_2.*

Then the *max* derivation mode can be defined as follws:

Definition 2.11. *The derivation mode of a P system is maximally parallel (denoted by max) if the residual object multisets in each membrane induced by applying (neglecting objects to be produced) a set of applicable rule multisets determined in line with max are dominated by object multiset on the left-hand-side (obm_l) of every rule in the P system.*

2.3 Reduction the computation of P systems to multi-criteria optimization

The computation of the set $Applicable(\Pi, C, \delta)$ can be expressed in terms of a multi-criteria optimization problem. The advantage of such reduction is an extensive standard vocabulary and a plethora of solving methods existing in the optimization area. This will allow to express different used algorithms and methods in a common language for a better comparison.

2.3.1 Preliminaries

A *multi-criteria* optimization problem (MCOP) is an optimization problem that involves multiple objective functions. In mathematical terms it can be stated as

$$\begin{aligned} &\max(f_1(x), f_2(x), \ldots, f_n(x)) \\ &\text{subject to } x \in X \end{aligned}$$

where $n \geq 2$ and X is the set of *feasible* vectors (or solutions). This set is usually defined by some constraint functions. One can also consider the objective function as a vector: $f : X \to \mathbb{R}^n$, $f = (f_1, \ldots, f_n)$. For a feasible solution x, the vector $z = f(x)$ is called an objective vector or an outcome.

When corresponding functions as well as f_i, $1 \leq i \leq n$ are linear, it is speaking about a multi-criteria *linear* optimization problem. It is also pointed that as for classical optimization problems the objective functions are minimized. The other

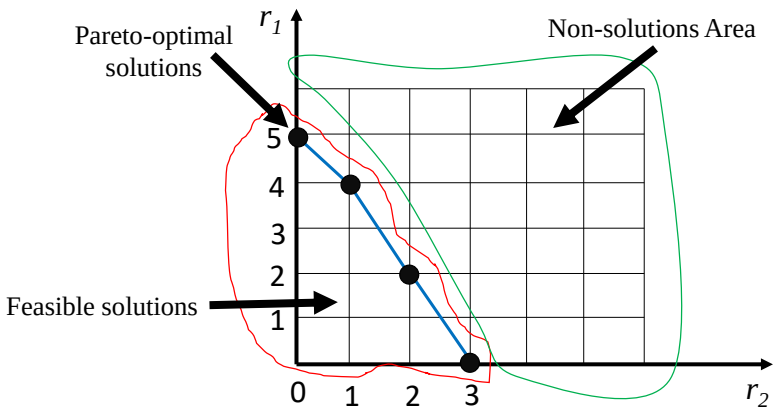


FIGURE 2.5: Feasible and Pareto-optimal solutions from Example 2.13.

cases like maximization or hybrid min/max can be easily reduced to the minimization one. When further $X \subseteq N^k$, $k > 0$ and $f : X \to N^n$, it becomes an *integer* multi-criteria linear optimization problem (IMCLOP).

In multi-criteria optimization, typically there is no feasible solution that minimizes all objective functions simultaneously. Hence, the main attention is focused on solutions that cannot be improved in any of the objectives without degrading some other objective(s). Such solutions are called *Pareto-optimal*. Formally, they are defined as pre-images of maximal elements of the outcomes, which are also called *Pareto front*.

Definition 2.12. *A vector $x \in X$ is Pareto-optimal for a MCOP (defined as above) iff there is no other vector $y \in X$ for which $f(x) < f(y)$, where $u < v$ iff $u_i \leq v_i$, $1 \leq i \leq k$, and $\exists j, 1 \leq j \leq m$ such that $u_j < v_j$.*

Example 2.13. *Consider the following problem:*

$$\begin{aligned} &\max(r_1, r_2) \text{ subject to} \\ &r_1 \leq 5 \\ &r_1 + 2r_2 \leq 6 \\ &r_2 \leq 3 \\ &r_1 \in N, \quad r_2 \in N \end{aligned}$$

The corresponding feasible solutions and Pareto-optimal solutions are shown in Fig. 2.5. As can be seen, the Pareto-optimal solutions are (5,0), (4,1), (2,2) and (0,3).

2.3.2 Rule choice as integer multi-criteria linear optimization problem

It is not difficult to see that the problem of the computation of elements from $Applicable(\Pi, C, max)$ can be reduced to IMCLOP. For the first time it was noticed in [97], but without any further development. For simplicity, consider that Π has only one membrane (and no environment). If this is not the case, apply the flattening procedure reducing it to one membrane [91, 94]. So $\Pi = (O, w_1, R)$.

Let $R = \{r_1, \ldots, r_n\}$ and $O = \{ob_1, \ldots, ob_m\}$. Consider that $r_i : u_i \to v_i, 1 \leq i \leq n$. Let C be the current configuration and let $C_{ob} = |C|_{ob}, ob \in O$.

Consider a set of variables x_i, $1 \leq i \leq n$ that indicate the cardinality of corresponding rules in some rule multiset $\mathcal{M} \in Applicable(\Pi, C, max)$, $|\mathcal{M}|_{r_i} = x_i$,

$1 \leq i \leq m$. Then the feasible set X of (asynchronous) solutions is defined by the following inequalities:

$$\sum_{i=1}^{m} |u_i|_{ob} x_i \leq C_{ob}, \quad \forall ob \in O, \tag{2.1a}$$

$$x_i \in \mathbb{N}, \quad 1 \leq i \leq n, \tag{2.1b}$$

Inequalities (2.1a) state that the sum of all consumed objects is included in C. Technically, for each object $a \in O$ it is verified that the weighted sum of rule instances and number of a in the left-hand-side of the rule is smaller than or equal the number of objects a in C. It is remarked that the system (2.1) can be also seen as a system of Diophantine equations and corresponding solutions are exactly describing the set of feasible solutions X.

The IMCLOP corresponding to the computation of $Applicable(\Pi, C, max)$ can be defined by:

$$\begin{aligned} &\max(x_1, \ldots, x_n) \\ &\text{subject to } (x_1, \ldots, x_n) \in X \end{aligned} \tag{2.2}$$

It should be clear that Pareto-optimal solutions represent exactly the multiplicities of rules for some maximally parallel solution.

Example 2.14. *Consider the system $\Pi = (O, w_1, R)$, with $O = \{a, b, c\}$ and $R = \{r_1 : ab \to abc; r_2 : bbc \to abb\}$. Consider the configuration $C = a^5b^6c^3$. Then, the constructed IMCLOP corresponds to the one given in Example 2.13.*

Therefore this system has 4 Pareto-optimal solutions: $(5,0), (4,1), (3,3)$ and $(0,3)$, corresponding to multisets of rules r_1^5, $r_1^4 r_2$, $r_1^2 r_2^2$ and r_2^3, which are exactly the maximal multisets of rules applicable to C. One of the solution can be obtained after solving the corresponding IMCLOP.

2.3.3 Tentative solutions

One of the traditional approaches to solve multi-criteria optimization problems is called *scalarization*. It consists in reducing the corresponding MCOP to a single objective optimization problem by using a real-valued scalarizing function involving the objective functions and additional scalar or vector parameters and variables. This can also imply additional restrictions to the feasible set based on the newly introduced variables.

One of the "simplest" methods to solve multi-criteria problems is the weighted sum method, where we solve the following single objective optimization problem

$$\max_{x \in X} \sum_{k=1}^{n} \lambda_k f_k(x) \tag{2.3}$$

The weighted sum problem (2.3) is constructed using the scalar product of the vector of objective functions f and the vector of non-negative weights $\lambda \in \mathbb{R}^n$ as a parameter. It is known that it allows to compute all Pareto-optimal solutions for convex problems by varying λ [98].

In the literature on P systems some simple variants of the weighted sum method can be found. In [99, 100] the vector $\lambda = (1, \ldots, 1)$ is considered (so the objective function is the sum of all variables). However, in this case only maximally parallel

rule sets having a maximal number of rules are obtained. In terms of the formal framework [91] this corresponds to $max_{rules}max$ derivation mode.

Another attempt was done in [101, 102] where the parameters λ_k correspond to the size of the left-hand-side of rules ($\lambda_k = |u_k|$, $r_k : u_k \to v_k$). Such optimization problem finds only maximally parallel solutions involving the maximal number of objects, corresponding to $max_{objects}max$ mode in terms of the formal framework.

In [103] the set of maximally parallel multisets of rules is expressed as solutions of a system of Diophantine equations (roughly equations (2.1a)) with an additional constraint to be satisfied on a solution, expressed as another system of Diophantine equations.

Finally, in [104] the set of maximally parallel multisets of rules can be expressed as solutions to a system of equations defining some Diophantine sets. While the construction is similar to the one to be given below, it is not trivial to manipulate Diophantine sets and it is not clear how to express the constraints as a single system of equations.

Below, a method named *Coefficient Non-deterministic Selection* (CNS) is proposed based on the weighted sum method and integer linear programming (ILP).

As above, consider that Π has only one membrane (and no environment). So $\Pi = (O, w_1, R)$. Let $R = \{r_1, \ldots, r_n\}$ and $O = \{a_1, \ldots, a_m\}$. Consider that $r_i : u_i \to v_i, 1 \leq i \leq n$. Let C be the current configuration and let $C_{ob} = |C|_{ob}, ob \in O$.

In the CNS method, $\lambda_{ob} = |u_k|_{ob}$, ob is one object in the left hand side of a rule. In addition to the λ_{ob}, introduce an integer parameter $M \in \mathbb{N}$ having a value that is sufficiently big (in fact it should be greater than the maximal possible value of any variable x_i, $1 \leq i \leq n$ of any feasible solution). Now modify the system (2.1) to construct a system of inequalities whose integer solutions will be maximally parallel multisets of rules for the configuration C.

$$\sum_{i=1}^{m} |u_i|_{ob} x_i \leq C_{ob}, \quad \forall ob \in O, x_i \in \mathbb{N}, 1 \leq i \leq n, \tag{2.4a}$$

$$\sum_{i=1}^{m} |u_i|_{ob} x_i + |u_k|_{ob} + M z_{ob}^k \geq C_{ob} + 1, \quad 1 \leq k \leq m, ob \in O, |u_k|_{ob} \geq 0, \tag{2.4b}$$

Inequalities (2.4a) are the same as (2.1a) and they state that the sum of all consumed objects is included in C. Inequalities (2.4b) state the maximality property of the rule set defined by $x_1, \ldots, x_n$. It verifies that for each rule there exist at least one object whose remaining quantity is not sufficient to apply this rule. They are based on multiple "either-or" constraints representation in ILP. The big value of M ensures that only one constraint from (2.4b) will be considered (the other ones will be satisfied because of the big value of M).

Hence, any solution $x_1, \ldots, x_n$ satisfying the system of inequalities (2.4) corresponds to a Pareto-optimal solution of (2.3), hence to a maximally parallel rule set $\mathcal{M} = r_1^{x_1} \ldots r_n^{x_n}$ applicable to configuration C. It is remarked that from the construction given above, it immediately follows that system (2.4) is Diophantine. The steps of CNS method are listed below.

1. Construct the coefficient matrix C_e according to the left hand side of all rules in the membrane, as shown in Equation (2.5);

2. Construct the system of inequalities corresponding to Equation (2.4a), as given in Equation (2.6);

3. Compute the all non-repetitive Cartesian products $\{c_{11}, \ldots, c_{1n}\} \times \cdots \times \{c_{m1}, \ldots, c_{mn}\}$. One Cartesian product is the vector of possible multiplicity of each object type in the left hand side of all rules;

4. Randomly select a Cartesian product as the value of $\{|u_1|_{ob_1}, \ldots, |u_m|_{ob_m}\}$;

5. For each element of C_e, compute $z_{ij} = sgn(c_{ij}), 1 \leq i \leq m, 1 \leq j \leq n$. Then construct the matrix Z shown in Equation (2.7). $sgn()$ is the sign function which is given in Equation (2.9);

6. Construct the polynomial matrix in line with the left hand side of Equation (2.4b), as shown in Equation (2.8);

7. For each row of Equation (2.8), choose polynomials without M to construct Equation (2.10) according to Equation (2.4b). If a row has more than one polynomial having no M, select the first one;

8. Combine Equation (2.6) and Equation (2.10) to calculate instances $x_1 \sim x_n$ of rules $r_1 \sim r_n$.

$$C_e = \left[\begin{array}{c|cccc} & r_1 & r_2 & \cdots & r_n \\ \hline ob_1 & c_{11} & c_{12} & \cdots & c_{1n} \\ ob_2 & c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ ob_m & c_{m1} & c_{m2} & \cdots & c_{mn} \end{array}\right] \tag{2.5}$$

$$\begin{bmatrix} c_{11} * x_1 + \cdots + c_{1n} * x_n \\ c_{21} * x_1 + \cdots + c_{2n} * x_n \\ \vdots \\ c_{m1} * x_1 + \cdots + c_{mn} * x_n \end{bmatrix} \leq \begin{bmatrix} C_{ob_1} \\ C_{ob_2} \\ \vdots \\ C_{ob_m} \end{bmatrix} \tag{2.6}$$

$$Z = \begin{bmatrix} sgn(c_{11}) & sgn(c_{12}) & \cdots & sgn(c_{1n}) \\ sgn(c_{21}) & sgn(c_{22}) & \cdots & sgn(c_{2n}) \\ \vdots & \vdots & \ddots & \vdots \\ sgn(c_{m1}) & sgn(c_{m2}) & \cdots & sgn(c_{mn}) \end{bmatrix} \tag{2.7}$$

$$S = \begin{bmatrix} c_{11} * x_1 + \cdots + c_{1n} * x_n + |u_1|_{ob_1} + M * z_{11} & \cdots & c_{11} * x_1 + \cdots + c_{1n} * x_n + |u_1|_{ob_1} + M * z_{1n} \\ c_{21} * x_1 + \cdots + c_{2n} * x_n + |u_1|_{ob_2} + M * z_{21} & \cdots & c_{21} * x_1 + \cdots + c_{2n} * x_n + |u_1|_{ob_2} + M * z_{2n} \\ \vdots & \vdots & \vdots \\ c_{m1} * x_1 + \cdots + c_{mn} * x_n + |u_1|_{ob_m} + M * z_{m1} & \cdots & c_{m1} * x_1 + \cdots + c_{mn} * x_n + |u_1|_{ob_m} + M * z_{mn} \end{bmatrix} \tag{2.8}$$

$$sgn(x) = \begin{cases} 1, & x > 0 \\ 0, & x = 0 \end{cases} \tag{2.9}$$

$$\begin{bmatrix} c_{11} * x_1 + \cdots + c_{1n} * x_n + |u_1|_{ob_1} \\ c_{21} * x_1 + \cdots + c_{2n} * x_n + |u_1|_{ob_2} \\ \vdots \\ c_{m1} * x_1 + \cdots + c_{mn} * x_n + |u_1|_{ob_m} \end{bmatrix} \geq \begin{bmatrix} C_{ob_1} + 1 \\ C_{ob_2} + 1 \\ \vdots \\ C_{ob_m} + 1 \end{bmatrix} \tag{2.10}$$

For some value of $\{|u_1|_{ob_1}, \ldots, |u_m|_{ob_m}\}$, there may be no solution after combining Equation (2.6) and (2.10). At this time, randomly selecting another Cartesian product

may result to a solution. When substituting all the Cartesian products one by one, the whole of solutions can be obtained. The CNS method asks for that coefficient matrix C_e must have zero elements, or M cannot be canceled and no solutions can be computed.

Example 2.15. *Consider the system* $\Pi = (O, w_1, R)$, *with* $O = \{a, b, c\}$ *and* $R = \{r_1 : abc \to ab; r_2 : a \to bb; r_3 : b \to cb\}$. *Consider the configuration* $C = a^2b^3c^2$. *One applicable rule multiset can be calculated in line with the proposed CNS method as follows.*

At first we can construct the coefficient matrix C_e,

$$C_e = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

According to Equation (2.4a), *we have*

$$\begin{bmatrix} x_1 + x_2 \\ x_1 + x_3 \\ x_1 \end{bmatrix} \leq \begin{bmatrix} 2 \\ 3 \\ 2 \end{bmatrix} \tag{2.11}$$

From C, $|u_k|_a \in \{1, 1, 0\}$, $|u_k|_b \in \{1, 0, 1\}$, $|u_k|_c \in \{1, 0, 0\}$. *So* $\{|u_k|_a, |u_k|_b, |u_k|_c\} = \{1, 0, 1\} \times \{1, 0, 1\} \times \{1, 0, 0\} = (\{1, 1, 1\}, \{1, 1, 0\}, \{1, 0, 1\}, \{1, 0, 0\}, \{0, 1, 1\}, \{0, 1, 0\}, \{0, 0, 1\}, \{0, 0, 0\})$. *Assume that the randomly selected Cartesian product is* $\{|u_k|_a, |u_k|_b, |u_k|_c\} = \{1, 1, 1\}$. *Because all the non-zero elements in C are ones, matrix Z is the same as C. From the left hand side of Equation* (2.4b), *we have*

$$S = \begin{bmatrix} x_1 + x_2 + 1 + M & x_1 + x_2 + 1 + M & x_1 + x_2 + 1 \\ x_1 + x_3 + 1 + M & x_1 + x_3 + 1 & x_1 + x_2 + 1 + M \\ x_1 + 1 + M & x_1 + 1 & x_1 + 1 \end{bmatrix}$$

Pick out elements without M, a system of inequalities can be built according to Equation (2.4b),

$$\begin{bmatrix} x_1 + x_2 + 1 \\ x_1 + x_3 + 1 \\ x_1 + 1 \end{bmatrix} \geq \begin{bmatrix} 2 + 1 \\ 3 + 1 \\ 2 + 1 \end{bmatrix} \tag{2.12}$$

Combine Equation (2.11) *and* (2.12), *we can obtain Equation* (2.13),

$$\begin{cases} x_1 + x_2 = 2 \\ x_1 + x_3 = 3 \\ x_1 = 2 \end{cases} \tag{2.13}$$

Solving Equation (2.13), *the solution is* $(x_1, x_2, x_3) = (2, 0, 1)$. *Its corresponding applicable rule multiset is* $r_1^2 r_3$ *which can be verified that it is a true solution conforming to the max derivation mode. Selecting different values of* $\{|u_k|_a, |u_k|_b, |u_k|_c\}$ *will lead to different solutions. The random selection of the value of* $\{|u_k|_a, |u_k|_b, |u_k|_c\}$ *can emulate the non-determinism of P systems. Note that for some value, there may be no solution. For example, for* $\{|u_k|_a, |u_k|_b, |u_k|_c\} = \{1, 1, 0\}$, *no solution can be obtained.*

Note that all solutions of (2.4) might be tedious to obtain. For the simulation purposes, only one such solution is necessary.

As previously mentioned, using the weighted sum scalarization technique it is theoretically possible to reach any single point from the Pareto front by choosing

appropriate values of the parameter vector λ. However, by using equations (2.4b) it becomes much easier to chose corresponding parameters as the feasible set which is restricted only to Pareto-optimal values.

In the literature on P systems, there are other examples of the construction of a single Pareto-optimal solution having certain properties. The direct non-deterministic distribution (DND) algorithm introduced in [103] for the FPGA implementations is such an example which can be explained in view of equations (2.1). First a random rule permutation is computed (corresponding to a random permutation of variable indices). Next, during the forward stage a random value (bounded by the number of possible applications) for the number of each rule applications is taken. This corresponds to finding the values of x_i, satisfying the constraints (2.1a). Finally, during the backward stage, the frequency of each rule is increased until it cannot be applied anymore. This step corresponds to the elimination of the dominated, *i.e.* smaller, solutions for the system (2.1) yielding only Pareto-optimal ones.

The described procedure can also be seen as a combination of scalarizing and of the lexicographic method, which is a method from the family of a priori methods for multi-criteria optimization. More precisely, it corresponds to a solution of a series of optimization problems, each of them bounded by the parameter corresponding to the choice of the maximal rule multiplicity for the forward stage. The backward stage also corresponds to a series of optimization problems that just reach the maximum for each component (like in the lexicographical method).

A simpler variant of the DND algorithm that does not perform the initial rule permutation can be found in [105, 106, 107]. It is remarked that the algorithm from [90] uses a different approach. It supposes that for a P system Π working in the derivation mode δ there exists a function $NBVariants(\Pi, C, \delta)$ that for any configuration C gives the number of solutions of (2.4). Next, it also supposes that there exists a function $Variant(\Pi, C, \delta, n)$ that for each integer n (up to the corresponding value) yields the corresponding solution (the used method is similar to the decoding of a number in the combinatorial number system).

2.4 Wrap-up

Core definitions and concepts of this thesis are presented in this chapter, including reaction systems and (symbol object cell-like) P systems. As a type of qualitative model, reactions in RS have inhibitors and duration limits. These two restrictive conditions plus the context sequence C_i control the transition of interactive process. Rules in P systems do not possess inhibitors, as a type of quantitative model, they are "constrained" by quantities of objects automatically. The interactive process transition of reaction systems and configuration transition of P systems are similar to some extend since they are driven by set of applicable reactions/rules. The computation of applicable rule sets of P systems is reduced to integer multi-criteria linear optimization problem (IMCLOP) to elucidate it. A tentative approach is proposed to address the IMCLOP.

Chapter 3

Overview of hardware implementation of reaction systems and P systems

For the sake of better explaining and understanding the work of this topic, the concept of *simulation* and *implementation* should be clearly distinguished. As a general purpose processing platform, the operation of CPU is not transparent to users. When a CPU is used to process parallel computing models, it cannot be guaranteed that the CPU strictly follows the theoretical calculation procedures of the model. Especially when processing large scale models, the parallelism provided by the current CPU multi-core architecture may not be enough to meet the practical needs. At this time, CPU can only turn the parallel operation into serial operation, although correct results can also be obtained. However, its calculation process is different from that of the model, and the parallelism is lost to some extent. This type of process is called *software simulation*. On the other hand, if the CPU or GPU has powerful parallel processing capability and can process models in exactly the parallel computing mode, the parallelism of the model will not be damaged. This kind of process is called *software implementation*. Computer software must rely on hardware to complete its work. From the perspective of hardware, software simulation or implementation is actually performed by CPU or GPU according to procedures defined by programs.

In effect, integrated circuits in processors execute operations in the bottom, rather than programs. So it is desirable to design tailored parallel circuits to carry out operations of a parallel computing model. This treatment is termed as *hardware implementation*. If such circuits are manufactured, an application-specific integrated circuits (ASIC) chip is obtained. Hardware implementation in this thesis emphasizes the design of circuits to process biologically inspired models. CPU/GPU simulation/implementation does not involving hardware circuit designs, for software development languages such as C++, Java and other high-level languages are used to develop corresponding programs to process reaction systems or P systems. Parallel architectures inside CPU/GPU have been manufactured and cannot be reconfigured. However, the hardware implementation uses hardware description languages (HDL) to design circuits, requiring the knowledge of digital circuit design and hardware architecture.

Because of the parallel processing capability of GPU and FPGA, they are used to simulate the parallel computing models generally. At present, there have been researches on GPU-based reaction system software implementations and FPGA-based P system hardware implementations. To better understand these implementation methods, it is necessary to take a closer look at both devices.

3.1 GPU presentation

Nowadays, multi-core architecture CPU is the mainstream. Whereas the component integration scale of some High-end GPUs with many-core architecture has outpaced the CPUs for the booming demand of graphics processing (advanced rendering and 3D vision) [108]. Different from FPGAs, there are manufactured parallel architectures in GPUs. The advantage is that developers should just concern about the efficient utilization of these architectures and the drawback is that these frameworks are un-reconfigurable.

Nevertheless, the GPU is not a general processing unit which can handle other computing assignments except for graphics processing. The predicament has changed for the arise of *compute unified device architecture*, known as *CUDA*, from the leading chip vendor-NVIDIA corporation. CUDA is a technology that enables *general-purpose computing on graphics processing units* (GPGPU). A CUDA-enabled GPU is an universal parallel computing device which is suitable for the implementing of parallel algorithm models. The parallel computing behavior of CUDA is based on the execution of multiple compute *kernels* on the GPU. These compute kernels are without physical construction, but based on an abstract parallel programming model. In other words, CUDA does not alter the physical structure of GPU. CUDA programming model is based on *heterogeneous computing*, where the CPU (*host*) is the master node that controls the execution flow and launches kernels on the GPU (*device*) when massive parallelism is required [109]. A kernel is executed by a *grid* of (thousands of) *threads*. The grid is a two-level hierarchy, where *threads* are arranged into *thread blocks* of equal size. Each block and each thread is unequivocally identified by an identifier. In this way, threads and blocks can be distributed easily to different portions of data, or to compute different instructions. Threads from the same block can be synchronized using *barriers*, while those belonging to different blocks can only be synchronized by the end of the execution of the kernel.

A GPU contains a *global memory*, which has the biggest size, but has the longest access time and a *shared memory*, which is smaller but faster [108]. Although current GPUs contain cache memories, in order to accelerate memory accesses, best performance is achieved when doing it manually. Global memory is accessed by all threads launched in all grids, and also by the host, but shared memory is only accessible by threads in a block. Threads also have fast access to their own registers for single variables, and local memory (which is normally outsourced to global memory). Accesses to memory have to be carefully programmed, so that contiguous portion of data is read by consecutive threads (providing so called *coalesced access*), since this increases the memory bandwidth utilization.

Now the architecture of GPUs is upgraded to *Streaming Multiprocessors (SMs)* which are composed of an array of *Streaming Processors (SPs)*, working as computing cores. A thread set consisting of 32 threads named *warp* is the basic unit which a SM performs its executions. A SM can manage multiple warps which are based on *Single-Instruction Multiple-Thread (SIMT)* model in effect. Each thread in a warp should commence its processing at the identical program address concurrently, although after beginning, threads can execute independently abiding by a sequential manner. The parallelism of CUDA is terminated when a warp branches or the memory stalls [111]. The SM framework and its warp flow is shown in Figure 3.2.

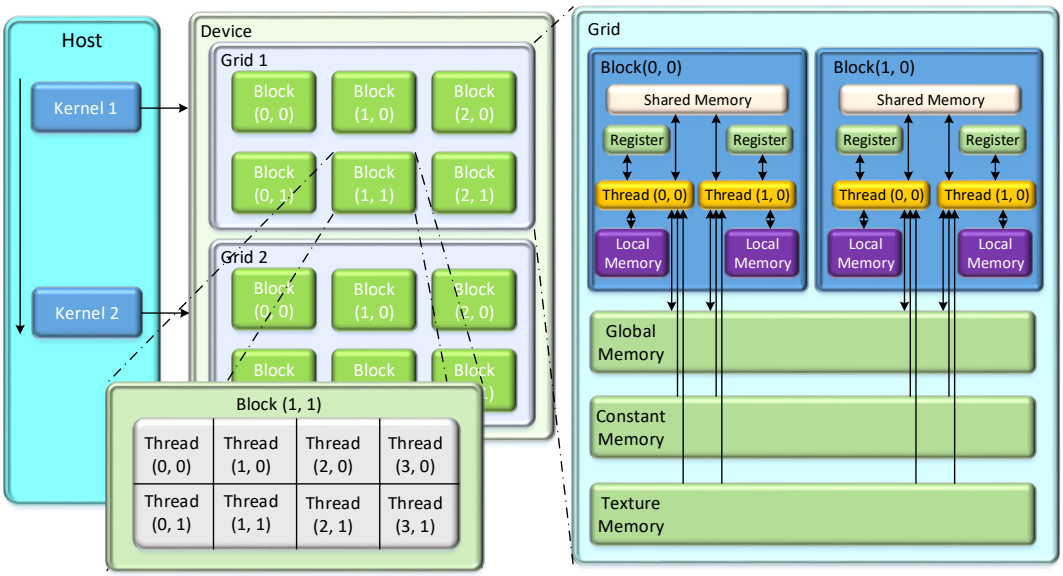


FIGURE 3.1: (left) CUDA execution model, (right) CUDA memory model [110].

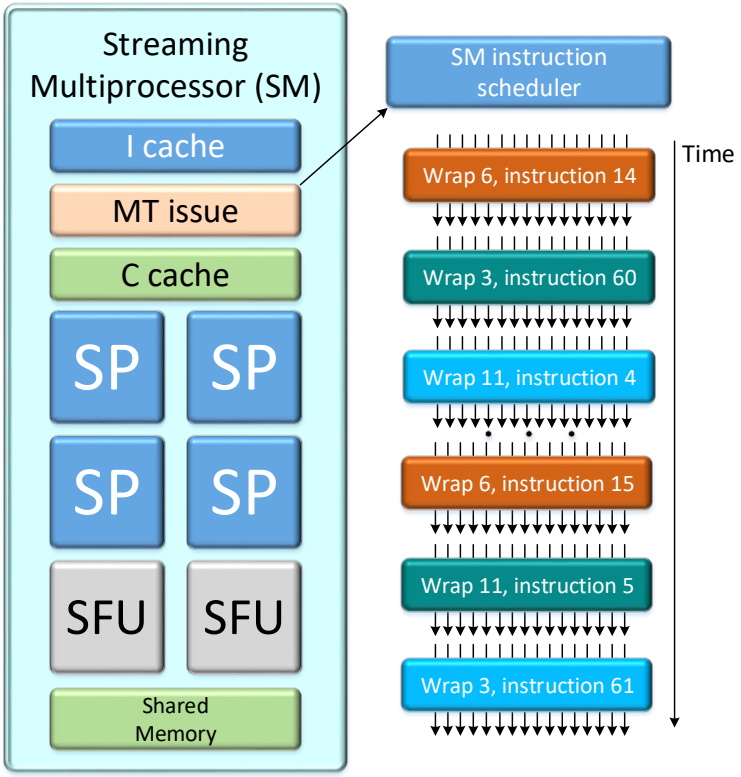


FIGURE 3.2: A SM is composed of an array of SPs, shared memory and a couple of caches. A Multiple-Thread tissue module contains a SM instruction scheduler which conducts instruction flows [112].

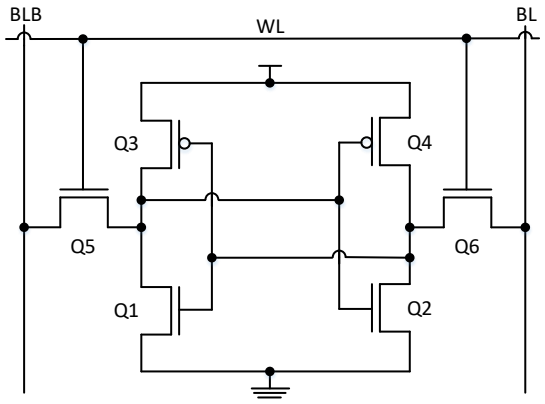


FIGURE 3.3: A 6T CMOS SRAM cell [115].

3.2 FPGA presentation

Filed programmable gate array (FPGA) is a type of reconfigurable integrated circuits stemming from programmable logic devices (PLDs). Programmable read-only memory (PROM) that came into being at 1970 [113] is regarded as the first PLD. From this simple device, it takes 14 years to evolve to the FPGA, when Xilinx Inc. invented such devices in 1984.

Classifying according to reconfigurable techniques underlying programmability of FPGAs, there are anti-fuse based FPGA, static random access memory (SRAM) based FPGA, electrical erasable programmable read-only memory (EEPROM) based and FLASH (which is a derivative of erasable programmable read-only memory (EPROM) with fast erasing process) based FPGA. Details of these technologies can be found in [114]. SRAM based FPGAs are the mainstream, thus this technology is revealed in detail.

3.2.1 FPGA architecture

The SRAM is a memory which can be constructed by 4 transistors (called 4T SRAM) or 6 transistors (termed 6T SRAM). A 6T complementary metal oxide semiconductor based SRAM is diagrammatically presented in Figure 3.3. The main upsides of SRAMs are their fast programmability, no limitation of reconfiguration times, apart from they are manufactured by the same CMOS technology employed to fabricate integrated circuits (ICs). The fact that a SRAM is composed of 4 or 6 transistors indicates that it occupies large chip area. This adversity is exacerbated for SRAM's volatility, which means data stored inside is lost when the power is cut off. To reprogram it after the host system is booted, an on-board microprocessor or external memory device which will consume more chip areas is required. These negative facets comprise the downside of SRAMs.

Different FPGA purveyors devised different architectures and components of FPGA. What makes things more confusing is that these components have unique names, although their physical structures and functions are similar. As can imagine, terminologies for FPGA of one vendor cannot be understood well for people who use products from another company. So it is difficult to portray FPGA from a general point of view. Xilinx FPGA architecture is exemplified to expound its internal structures for I employ FPGAs purchased from this company. The basic component of Xilinx FPGAs is entailed as *logic cell*, which is arranged in a way named *look-up table* (LUT). Take a Boolean expression $y = (\bar{a} \wedge b) \vee \bar{c}$ as an example to explain how

TABLE 3.1: Truth table of $y = (\bar{a} \wedge b) \vee \bar{c}$.

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

a LUT in logic cells works. The truth table of this expression is shown in Table 3.1. The 8 binary values in column y are programmed in 8 SRAMs when $y = (\bar{a} \wedge b) \vee \bar{c}$ is detected to be executed. A storage element is needed to hold state [66] in the circumstance of designing sequential circuits where clock signal is indispensable. As a consequence, a D flip-flop is added besides the column of 8 SRAMs. Which value will be output is determined by the input combinations of abc. A 8-input-1-out multiplexer is used to output the result according to inputs. If the function of a Boolean expression changes, just reprogram values in the 8 SRAMs to redefine it. Typically, 2 logic cells are assembled in a higher hierarchy called *slice*, which is illustrated in Figure 3.4.

This construction of a cell is termed as 3-input LUT (3-LUT for short hereinafter) in which results are calculated by consulting a truth table associated with the desired function with 3 input variables. The number of input ports of a LUT varies from 3 to 6. As a memory device, a SRAM based 3-LUT is versatile to be used as 8×1 bit RAM or 8-bit shift register (SR). The reconfigurability of a FPGA primarily originates from the programmability of interconnects wiring different slices. Programmable connections and switches are organized in particular blocks linking to output of multiplexers of logic cells. These combinations of connections can build highly interconnected wires, as captured in Fifure 3.5, to carry out sophisticated functionality/algorithms.

In digital electronics, a clock is a signal oscillating between high and low electrical level. In low-cost electronic product like an ordinary micro-controller, clock signal, commonly called as clock, is generated by a resistance capacitance (RC) oscillator [116, 117]. While for high-end IC products like an FPGA, a clock generally starts off with a quartz crystal resonator [118], which is a small slice of quartz crystal combined with integral amplifier circuits [119]. The oscillating frequency of a quartz crystal is determined by the shape and size of the crystal slice. The main advantage of this quartz crystal is its tolerability to temperature variation, outputting more stable frequency when temperature highs and lows comparing to RC oscillator.

The importance of clocks is that their signal is employed as a kind of metronome to trigger so to synchronize operations in circuits. There are two types of trigger modes in digital circuits: edge trigger and level trigger. Edge means the transition from high level to low level and vice versa, corresponding to falling edge and rising edge respectively. Level trigger is more intelligible that operations are executed when the voltage is high level or low level. The time interval between two rising (or falling) edge is called clock cycle, which is a time constant numbered by the previous rising edge's sequence number. Then the clock cycle adjacent before and after n*th* rising edge are the (n-1)*th* and n*th* time cycle. It is highlighted here that clock rising edge is used as trigger signal in researches of this thesis.

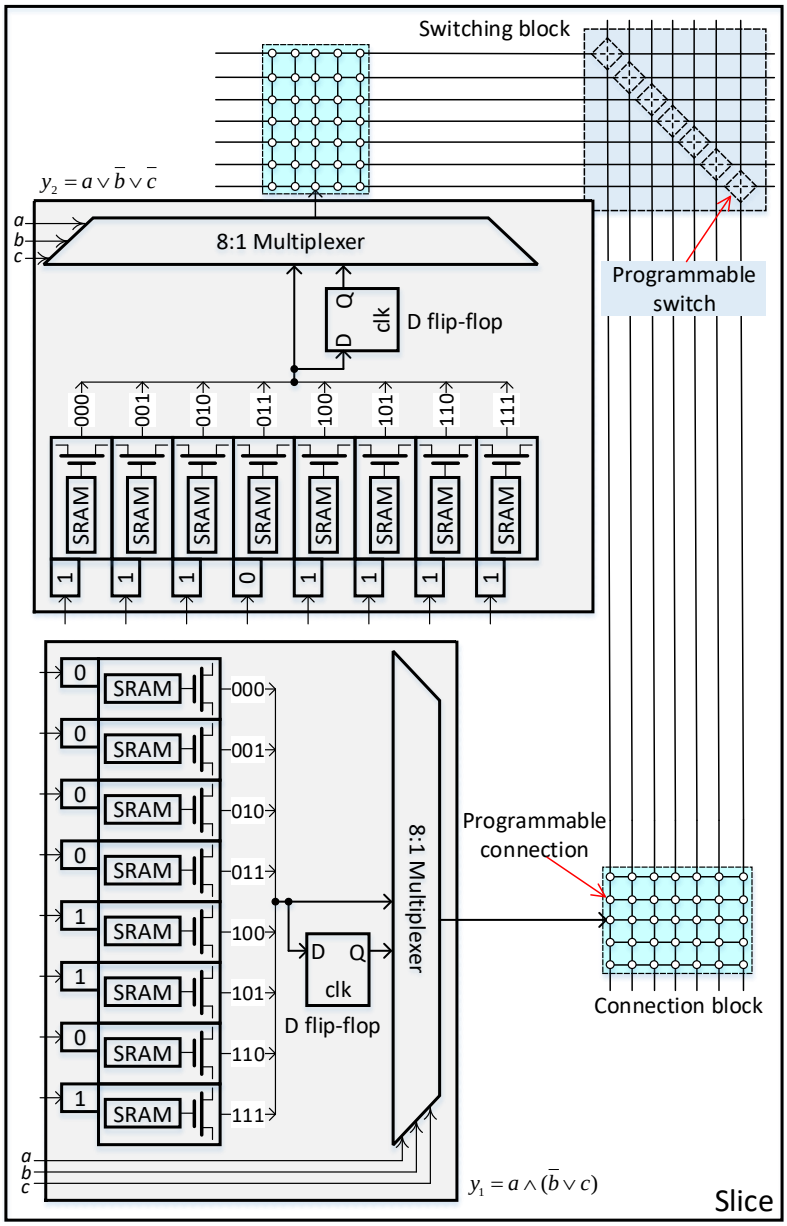


FIGURE 3.4: A slice containing 2 logic cells. These 2 logical cells perform $y_1 = a \wedge (\bar{b} \vee c)$ and $y_2 = a \vee \bar{b} \vee \bar{c}$ respectively.

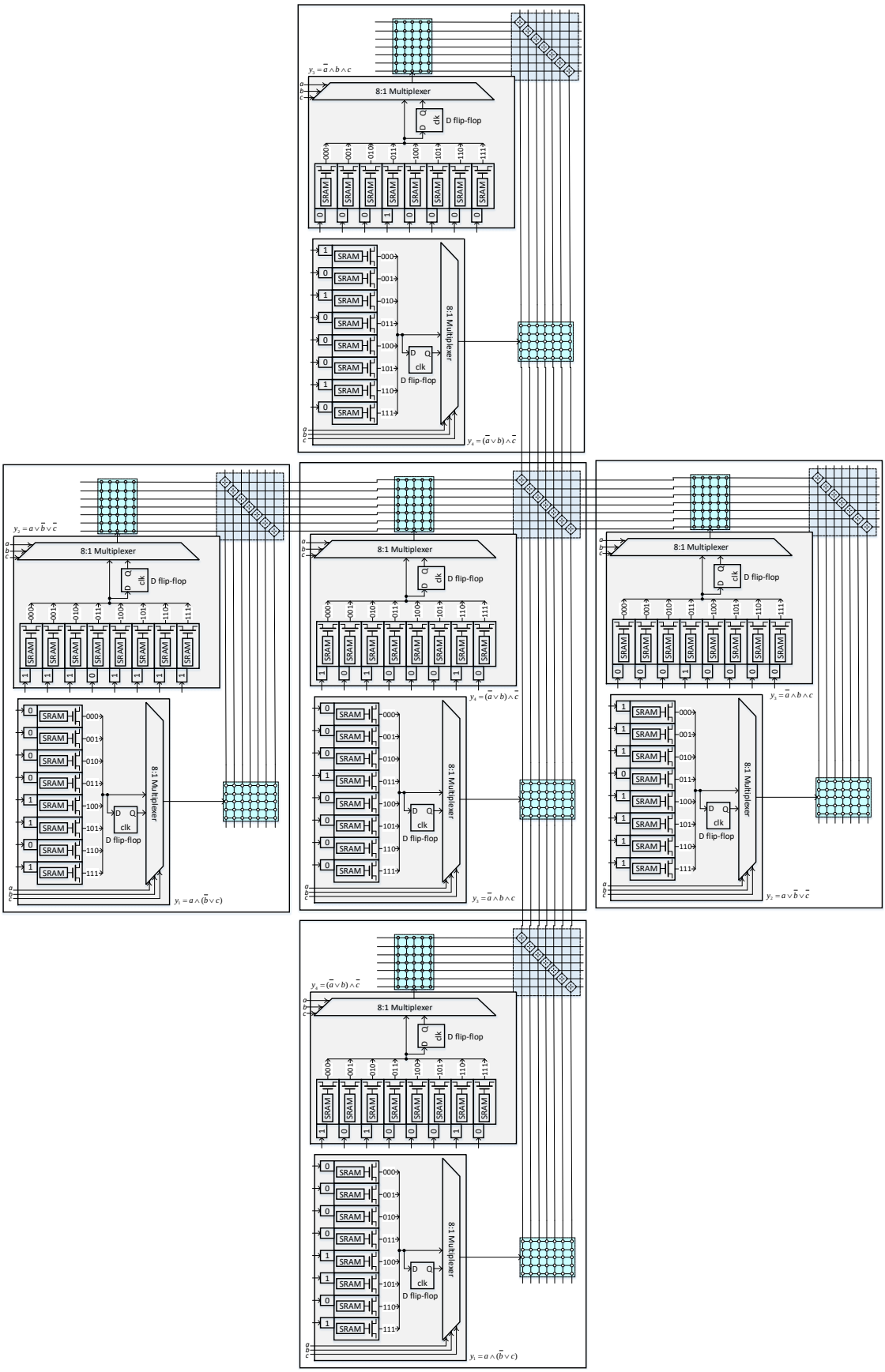


FIGURE 3.5: The interconnection of 5 slices. Connection blocks (CBs) and switching blocks (SBs) impart highly flexibility for FPGAs to reconfigure their architectures to perform various functions. The CB and SB can contain much more intersection points than that depicted in this figure.

Synchronous elements having clock ports such as flip-flop can be connected to the same clock. If this clock is shared by a plenty of synchronous elements, it is branched many times, resulting a clock tree [113]. All these elements connected to the clock tree are triggered simultaneously if they are activated by the same clock edge. This is the source of the massively parallel processing ability of FPGAs.

3.2.2 FPGA development flow

With the digital circuit scale increases rapidly after the invention of transistors in Bell laboratory in 1947 [120], and as the scale of digital circuits increase dramatically, obeying the Moore's laws loosely in the past decades, drawing the schematic to design large scale circuits is infeasible. Characterizing the functionality of circuitry with hardware description language (HDL) then compiling this characterization to schematic is the core task of electronic design automation (EDA). With EDA technology, designing large scale digital circuits in a relatively short period had became reality.

Digital systems consist of combinatorial and sequential logic components which contain registers, clocks and their control mechanisms. For example, a flip-flop comprises a register which can perform elementary operations including load, count and shift operations. *Register transfer operations* refer to operations aiming at data stored in registers [121]. If a digital system is designed by registers, involving register transfer operations and control procedures to these operations, then the digital system is illustrated at *register transfer level* and a register transfer level model is obtained consequently [122, 123, 124]. Register transfer level has a higher abstraction than gate level, which specifies models in the form of schematics.

FPGA development flow commences from RTL model design, ending up with results checking performed by integrated logic analyzer or oscilloscope. This process varies with different developing software provided by different FPGA purveyors. As Xilinx FPGAs are selected to implement computing models, the developing software is Vivado [125] FPGA integrated development environment (IDE) which is a new generation of IDE released by Xilinx since 2012. Vivado based development flow [126] is expatiated as follows. Other IDE based development processes provided by different vendors are similar in essentials while differing in minor points.

RTL model design [127]. RTL models are specified by HDL such as VHDL and Verilog specification languages. Verilog is used to design RTLs in this thesis. Verilog, initiated by Phil Moorby in 1985, is a IEEE standard HDL since 1995. The basic functional unit in Verilog is *module*. The functionality of a system is distributed among nested modules since a module can instantiate others modules to incorporate them [128]. A module is a functional block with input and output ports through which it receives data and sends outcomes from/to other modules.

RTL models should be verified to meet design requirements. The validation procedure is called *functional simulation*. In Verilog, a special module named *testbench* is designed to validate RTL models. In such a module, a target RTL model is instantiated and imposed specific input signals. The outcome of testbench in form of waveform or plain text tells the accuracy of this model. It is remarked that testbench just performs software simulation of RTL models, which means the execution of RTL model is conducted by CPU of host computer rather than on the adopted FPGA. Functional simulation is also called *behavioral simulation* [129, 130]. This simulation neglects the latency when signals pass through logic gates so that results are obtained at the trigger time, i.e., at the rising edge of clock. It is not the case

when variables are processed by real digital circuits in FPGA where operations are triggered by clock edges and complete after some time interval.

Synthesis [131]. The impractical of designing large scale circuits with schematics does not mean that schematics are not important. In fact, schematics of circuits are the ultimate goal of design specification phase for ICs and ASICs. For FPGA implementation, the corresponding schematics are needed as well for programmable logic blocks that are configured to perform the functions of these schematics, although the actual the layout in a FPGA is not the circuits. After verification of circuits, the schematics will be used to manufacture hardware integrated circuits in silicon wafers. A compilation process from register transfer level to schematic level termed *synthesis* is responsible for the automatically drawing of schematics. RTL models are reliably synthesizable inputs for synthesis tools supplied by a variety of vendors.

Setting up physical constraints and timing constraints [132, 133] are the two subsequent procedures after synthesizing RTL models. Physical constraints assign ports in RTL model to FPGA pins so that signal input & output course can be carried out on real circuits. Timing constraints set the period of the clock source (a RC oscillator or a quartz crystal resonator), and signal input & output delay so that a real clock with specified cycle will be produced. The package view of target FPGA which list all the available pins can be opened under *I/O Planning* view of Vivado. For vector ports (more than one bit), each bit should be assigned a pin. Clock signal is a 1-bit scalar which demands one pin, while a differential clock requires two pins to accommodate 2 differential ports.

Results computed by FPGA are not shown straightforwardly like software simulations, but involve a technical procedure named *hardware debug* to examine. Hardware debug procedure can penetrate the results stored in registers of FPGA. Variables to be debugged should be marked in the RTL code or in the netlist generated after synthesis. Then it is needed to set up debug cores in the synthesized model. Vivado will probe these marked signals and show their values via integrated logical analyzer (ILA), a virtual software logical analyzer. Alternatively, it is possible to output the results from a pin and then analyze using an oscilloscope. Or send results with some communication method, like serial port communication, to the host computer to display. But much more efforts are involved by this way.

Implementation [134]. Several optimizations and important tasks such as physical synthesis, static timing analysis and layout (place & route) are performed during implementation step, as shown in Figure 3.6. The good news is that these operations are executed by Vivado automatically. A important simulation after implementing is post implementation timing simulation which reflects the real elapsed time of physical circuit running since the time delay of logic elements and data paths are taken into account. Consequently, performance of a design can only be revealed by post implementation timing simulation. It might also be noted that only Verilog HDL is supported by Vivado to do post implementation timing simulation [135].

Generate bitstream, device program and hardware debug [136, 137] If post implementation timing simulation meets design requirements, FPGA configuration file, bitstream file, can be generated. Bitstream file contains all the design contents and will be downloaded to FPGA to program it, i.e., downloading bitstream file to configure device. After programing of target FPGA, the dashboard of integrated logical analyzer opens automatically. The marked variables should be added in to show

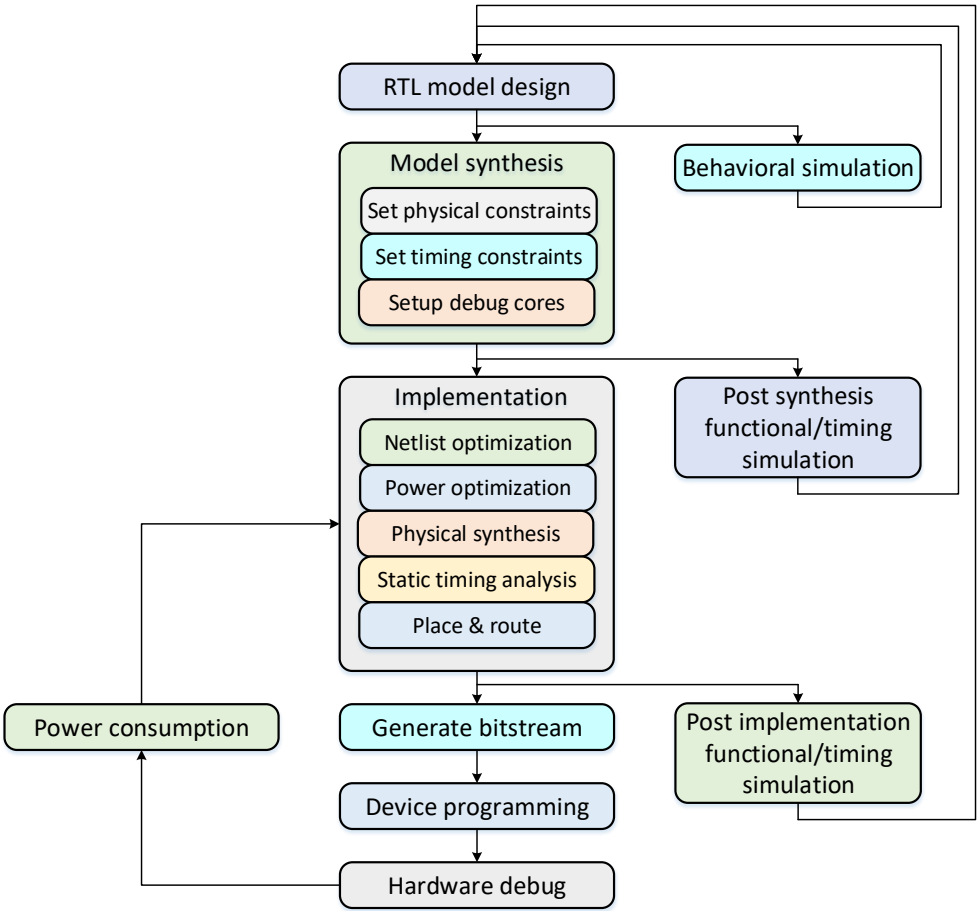


FIGURE 3.6: Vivado FPGA development flow.

their values obtained in hexadecimal form by FPGA. The whole development flow is given in Figure 3.6.

3.3 Software simulation and implementation of reaction systems: an overview

For the present, hardware implementation of reaction systems has not been reported. So only software simulation is reviewed. From the point of hardware, a software simulation is performed by the CPU of host computer. Compute unified device architecture (CUDA) enabled GPU [138, 109] implementations also belong to software simulation taking into account that GPUs are programmed by software languages like C++ rather than HDLs. Furthermore, hardware architectures inside a GPU are prefabricated which are not reconfigurable. Nowadays, CUDA-enabled GPUs are powerful processing platforms which execute myriad threads in parallel.

3.3.1 CPU simulation of reaction systems

A reaction system simulation engine `brsim` implemented in Haskell is developed in [139]. For its native compilation trait, the efficiency of Haskell is higher than languages needing interpretation, like Python. Based on this engine, a browser-based version `WEBRSIM` coded in PHP and JavaScript is proposed [140, 141]. In addition

to simulating RSs by inputting the set of reactions defining a concrete reaction system and corresponding context sequence C_i, `brsim` has the ability to output conserved sets [142], behavior graph and conservation dependency graph of the target RS. `brsim` supports bath mode in which the state sequence W_i going through is listed and step-by-step mode in which a single C_i can be input to drive RS and output consequential W_i. The source code of `brsim` is provided in [143].

Another untitled software tool implemented in C++ which verifies initialized context restricted reaction system (icrrs) and computation tree logic for reaction systems (rsCTL) properties is devised in [144]. This tool is designed based on monolithic/partitioned encoding mode and binary decision diagrams (BDDs), dealing with the transition relations of interactive process. The elapsed time/consumed memories–model size curves of four case studies are plotted to explicitly exhibit the tool performance. Experimental results show that monolithic encoding has better performance than partitioned encoding, although the latter one may has better memory efficiency.

3.3.2 GPU implementation of reaction systems

One GPU-based implementation of reaction systems written in CUDA, Highly Efficient REaction SYstem simulator (HERESY), is present in [145]. HERESY [146] has a CPU-based engine coded in Python to cope with simulations of small-scale RSs whose entities (reactants, inhibitors and products in reactions involved) are less than 100. The decision of choosing which engine to use is made by HERESY itself without manual intervention. The simulating process of HERESY is a iteration consisting of four steps. Reaction set, context sequence C_i and initial reactants are loaded in GPU in the first step which is performed by host CPU. Then add the context elements of the n-th step to the RS. Assume S is the background set, a number of $|S|$ threads are allocated to execute memory updating of $|S|$ reactants. In the third step, evaluate $T' = res_{\mathcal{R}}(T)$, where T is the initial state of target RS. At last, $|S|$ threads are employed to store the transition of interactive process. Bump step counter up by one until the predefined step number is reached.

It is a fine-grained parallel implementation of RS since every applicable reaction in each step is executed by distinct parallel threads of GPU. Reactions are saved in the constant memory instead of global memory so that the access latency declines to a large extend. To be specific, reactions are encoded as 3 sets of positive integers indicating R_r, I_r and P_r respectively. These sets are arranged into an one-dimension array to facilitate to be processed on GPU. Entities produced in state sequence W_i are output to represent the interactive process. The handy GUI of HERESY makes it easier to use for reactions and context sequence specification can be typed in different boxes in the upper part and interactive process is shown in the lower part of the GUI.

The correctness of HERESY is validated by simulating *lac* operon reaction system [147]. A serial of reaction systems whose sizes range from $|S| \times |A| \in (10 \times 10, 100 \times 100, 1000 \times 1000, 2000 \times 2000)$ are generated randomly to evaluate the performance of GPU-based HERESY and its speed-up comparing to `brsim` and CPU-based HERESY. The maximal speed-up of HERESY GPU/CPU and HERESY GPU/`brsim` achieved are $29.06\times$ and $5.19\times$ simulating the $|S| \times |A| = 2000 \times 2000$ RS.

HERESY is put into a practical application to simulate the ErbB receptor signal transduction in human mammary epithelial cells [148] which is modeled as a RS with 6720 reactions and 246 entities. Providing that the context sequence C_i drives

1000 step of interactive process, the speed-up of HERESY GPU/CPU is 26.28× while the speed-up of HERESY GPU/`brsim` is 10.97×.

3.4 Background of hardware implementations of P systems

Due to its historical background, P systems were used as a modeling framework for biological and ecological subjects. On the other hand, the inherent large scale parallelism of P systems has the profound potential for the progress of extreme data processing. Thus, an interesting topic is the implementation of P systems on contemporary silicon integrated circuits. This allows to exploit the desirable parallel computational capability of P systems to explore a new orientation for high performance computing (HPC) [149, 150].

The augmentation of electronic ingredients' density had been subject to the well-known Moore's law for decades. After extraordinary exponential growth of many years, the number of transistor in chips cannot follow this law, at least it cannot be doubled within two years [151, 152]. With transistors shrunk to nanoscale, quantum effects stand out [153], the behavior of circuits is not up to expectations. Another knotty problem is the heat dissipation, which would melt the silicon substrate with density level increasing. Traditional semiconductor scaling is predicted to reach an end by about 2024 on the foundation of prior arts [154]. Parallel computing has the potential to further uplift computing power provided that the density of transistor is constant [155] with multicore and multithread architecture, although heat dissipation and interconnect issues would be challenges [156].

Before claiming that the era of parallel computing has dawned, one essential question should be clarified: what does parallel computing mean? Though not rigorous, parallel computing implies computing based on decomposition of concurrently executable operations into some type of construct and assignment of operations to parallel processing nodes. The evolution of computer processor scheme, from single-core single CPU to multiple-core single CPU and multiple-core multiple-CPU frame, is a instance of parallel computing. In this sense, the inherent parallelism infers that P systems belong to parallel computing domain. What constituents work as processing nodes will give rise to different implementing strategies. The way how living cells allocate, organize and coordinate processing nodes has evolved for billions of years. Investigating this magnificent course will illuminate us to handle the multiple cores computing, which has cut a striking figure in the contemporary parallel computing realm.

The large scale distributed parallel process occurring in vesicle compartments and the vesicle division functionality enlightened from mitosis of living cells which increase the artificial cells exponentially are two of the most outstanding advantages of membrane computing that would underlie the foundation for the construction of highly parallel computation platform whose performance, flexibility and scalability outperforms traditional sequential counterparts substantially [157]. As a parallel computing paradigm inspired by the structural and functional features of biological cells/membranes, only the parallel computing platforms are suitable for the implementation of P systems with respect to the fact that the limited parallelism of general computers realized by the communication mechanism among the multiple cores of CPU cannot make full use of the large scale parallelism, non-determinism and other particular attributes that impart an enormous computing potential like creation and dissolution of inner membranes, self-replication or *autopoiesis* [158] of the whole cell-like entirety that works as a computing unit, the *symport* and *antiport* of objects, etc.

Implementing this new computing paradigm which exhibits a promising prospect on parallel platforms so that can utilize the excellent theoretical performance for practical applications is the purpose. It is remarked that programming the membrane computing algorithms with high level general purpose language and executing them on the computer is just simulating, not real implementing [46] of P systems.

Software-based and hardware-based parallel computing platforms had been developed for implementing P systems. A software-based parallel computing platform is constructed on a cluster of computers [159]. This platform achieves good performance and flexibility for which CPUs execute the operations and the changing of subjects carries out easily by programs. Nonetheless, with the size of target P system increasing, the time and CPU resources expenditure caused by the communications of different computers are rising dramatically. Moreover, the underlying hardware (a cluster of computers) of this platform cannot be miniaturized so that membrane computing algorithms cannot be utilized in embedded chips and compact controllers which can be employed in robots, automobiles, machine tools, etc. This disadvantage limits the range of applications of membrane computing. However, hardware-based platforms are fabricated on integrated circuits (mainly on FPGAs). In this case, P systems are mapped to digital circuits and the performance is much higher compared with the software-based platforms, although this high performance may come at the cost of flexibility and extensibility. But the reprogrammable hardware turns the corner, lifting barriers to devise portable and embedded membrane processors which can be used as CPUs, controllers or something like that. This is the necessity and importance of hardware implementation of P systems.

Hence, it is important to propose hardware implementations of P systems as specific architectures that do not have the drawbacks related to the traditional ways of implementation. The principal direction for such researches FPGA implementation in which new parallel circuits are specially designed to implement some variants of P systems.

The core problem for the implementations is the *object distribution problem* (ODP) that computes the multiset of applicable rules based on current configuration to transform it to the next. This problem is a particular variant of a more general problem that computes the applicable set of multisets of rules for a configuration and it is known to be NP-complete [101]. We recall that in the general case the model is nondeterministic, so an equitable choice among different possibilities should be provided. Known algorithms and heuristics do not parallelize well, so special heuristics were developed in order to quickly compute the desired multiset of rules.

3.5 General ideas about hardware implementations of P systems

The discussion about hardware implementation of P systems concerns mainly the simulation of concrete variants (sometimes even examples) of P systems using a dedicated hardware (FPGA). In most of the cases a (single) particular class of P systems is simulated. The simulator is composed from two parts: (1) the hardware simulator for a concrete system and (2) the software generator of hardware simulators, which based on input parameters (rules, membranes, initial configuration etc.) generates the code for the dedicated hardware simulator. This section mainly discusses the structure of the corresponding hardware simulators, as the generator part is more-or-less following standard compiler construction techniques. I will concentrate on three points, which are the most important for a hardware implementation:

(a) the representation of the configuration (multisets of objects and the membranes), (b) the representation of rules and their parallel application and (c) handling of the non-determinism.

3.5.1 Data organization

Membranes. There are no compartments in silicon circuits, hence the notion of membranes is relatively difficult to represent directly. Nevertheless, according to [46], a membrane is just an idealized concept without internal structures. The main functionality of membranes is to perform a topological division of the space allowing P systems to compute in a distributed manner (based on an correspondence between the membrane and its contents). Hence, the spatial placement and size of membrane are not important, only the inter-relationship among them matters. Moreover, it is known that any membrane structure can be reduced (flattened) to just a single membrane, see [91, 94] for more details. The existing hardware simulators adapt in most cases this last point of view, where the membrane structure is not physically implemented on the device. As examples from this rule we cite [160] and [161] (region-based), which implicitly implement the membrane topology by using dedicated buses and message passing in the corresponding circuits.

Configuration. The representation of the configuration in all cases is done as a vector of non-negative integers (stored in the memory/registers of the device). We remark that this vector corresponds to a flattened system, so it is relatively big and sparse, as it encodes each object/membrane pair. For performance reasons, it is physically split in several places to be closer to the processing units (as routing is relatively expensive). In the case of [160, 161] it can be argued that corresponding parts are internalized into the corresponding region circuit, as the access to corresponding values is not direct and it is done by message passing.

Evolution Rules. As it can be seen from definition 2.9 in the simple case (without permitting and forbidding) rules can be defined by 2 integer vectors indicating the multiplicities of corresponding objects in the left-hand-side and right-hand-side of each rule. This gives a natural rule representation as two vectors stored in the memory/registers. Then a specific circuit/module verifies the applicability of rules and performs their application. Most variants of hardware implementations use this idea, however in [160, 162, 163] each rule is encoded in hardware as a specific circuit that verifies the needed resources and performs the rule application.

3.5.2 Object Distribution Problem and Non-Determinism

The hardware implementation of P systems faces the problem of the computation of the applicable rule set according to some derivation mode (usually maximally parallel). More precisely, an efficient way to compute and represent an element from $Applicable(\Pi, C, \delta)$ is required [90, 164]. The difficulty of the problem is that rules can compete for same objects, so increasing the number of occurrences for one rule, may decrease the application possibilities for another one. Another important problem is to ensure that a non-deterministic choice among all possibilities is performed. In [165, 166, 167, 105], hardware architectures aiming at parallel processing and communication, and the application of rules are developed. In [168], a formal exposition of non-deterministic evolution in transition P systems was suggested.

The first problem is called as *object distribution problem* (ODP). It consists in the computation of the set $Applicable(\Pi, C, \delta)$ (or of an element from this set). As discussed in Section 2.3 in terms of multi-criteria optimization this corresponds to the computation of the corresponding Pareto front (or an element of it).

In [103] different algorithms solving ODP are classified in *direct* and *indirect* ones. In the direct approach, the corresponding multiset is directly constructed by the algorithm. In terms of MCOP this corresponds to a particular fixed scalarization. The indirect approaches are based on the observation that the solution number is finite, because the solution space is bounded by the size of the configuration. Hence, a heuristic or brute-force approach can be used to explore this bounded space. However, since it is an overestimation, there might be visited elements that are not valid solutions. Hence, the algorithms are iterative and explore the whole space until a valid solution is encountered. In terms of MCOP this corresponds to different searches through the space limited only by the maximal values for each axis.

Sometimes it is not easy to classify an algorithm in one of these categories. Here classify an algorithm as a direct approach if its main goal is to construct a valid multiset of rules. Otherwise, if an algorithm is exploring different solutions until it reaches a valid one, it will be classified as indirect. We will use this classification to overview different strategies for ODP solution known in the literature.

Indirect approaches. Generally, the enumeration of all possible solutions and their verification one by one until a correct solution is obtained is the simplest method for the indirect approach [169]. Before the first correct solution is obtained, some invalid solutions should be rejected. This approach is called *indirect straightforward approach* [103]. Taking into account that it is not viable to enumerate all possible solutions for many problems, the feasibility of the approach is low. However, the performance of the algorithm suggests its use as to compute the floor values for the object distribution problem. Another indirect approach discussed in [103, 161] called *indirect incremental approach* investigates a strategy generating possible solutions in rounds. Other attempts based on a similar idea but with different rule elimination strategies were done in [170, 171, 107, 172, 173, 174].

Direct approaches. In contrast to indirect approaches, the direct approach fabricates a solution straightforwardly rather than identifying a number of possible solutions before a solution is confirmed.

The simplest approach is the *direct straightforward approach*. As defined in the paper [103], in this approach "all the solutions to the object distribution problem are given as input, and one of these solutions is simply selected at random". While in the same paper it is argued that such approach is infeasible for an arbitrary configuration and rule types, it can still be applied in a big number of cases. As shown in [90, 164], if at each step the number of solutions can be expressed as a the number of words of some length in a regular language, then it becomes possible to compute the solution only based on its number. In [90] it is shown that the corresponding class of P systems is quite large and also that this method is particularly interesting for bounded derivation modes like the set-maximal derivation mode (called also flat mode) where the rules are chosen in a set-maximal way (instead of the multiset maximal way).

Another variant of the direct approach is the *Direct Non-deterministic Distribution algorithm* (DND) proposed in [103]. A similar algorithm can also be found in [175, 176]. This algorithm works in 2 phases. At the first phase all rules (initially randomly

shuffled) except one are selected to be applied a random number of times below its maximal applicability value. In the second phase, all the rules are taken in the converse order (with the first one being the rule excluded at the previous step) and their applicability is increased up to the maximal still possible value. A variant of DND, named DND-P, became popular in the simulation of Population Dynamics P (PDP) systems [177]. Together with another algorithm DCBA [178] it was employed for the engine of the PDP system simulator on CUDA [179].

Non-determinism. One of the difficulties of the above approaches is the handling of the non-determinism. From the formal point of view, the non-determinism corresponds to a random equiprobable choice of an element from the set of all applicable multisets of rules ($Applicable(\Pi, C, \delta)$). In the case of indirect approaches, due to the iterative nature of the algorithms, it is not easy to argue that each possibility has the same probability to occur. We would state that solutions containing a smaller number of different rules have a higher chance to be selected. In the case of DND algorithm and related variants, it looks like the obtained solution tends to be an equiprobable choice. However, the corresponding articles do not give such a proof and there are some unclear points, which do not allow us to affirm this fact. Up to now, the only algorithm that is performing a truly non-deterministic choice is the one described in [164, 90]. However the corresponding implementation is limited to some particular derivation modes and particular classes of P systems.

3.6 Literature review of existing P system FPGA implementations

With the advent of reconfigurable hardware which realizes the idea of modifying the hardware circuits by programming, conceiving a novel circuit simulating an innovative processing paradigm is no longer an exceedingly hard task. The first attempt to use FPGA reconfigurable hardware to simulate P systems dates back to 2003 [160]. Since then two simulation approaches emerged, considering regions or rules as basic processing units.

3.6.1 Region-based implementation

In the region-based simulation approach rules and objects from different membranes are physically located in different places of the circuit, while those from the same membrane are physically close and well connected. The biggest problem is to ensure the correct communication of objects between membranes as this requires a global level synchronization. As advantage, the obtained system is highly scalable and robust. Below, we give two examples of region-based simulations.

In contrast, the rule-based implementation approach discussed later explicitly represents the evolution rules as processing units and multisets of objects as register arrays, while membranes and regions are represented implicitly as logical constructions existing between those processing units and data structures.

Petreska and Teuscher implementation

As the spearhead of simulating membrane computing on FPGA, some groundbreaking matters had been devised by Petreska and Teuscher. For instance, trading communication for membrane containment relations, taking the priorities of rules into

account, proposing the first attempt to simulate membrane creation and dissolving mechanism in integrated circuits [160], etc. Their outstanding achievements inspired the successors to engage in this challenging and breathtaking field to advance the development of hardware simulation of P systems. For the sake of precision of the general model, a P system is modified in two aspects: the application of evolution rules in each membrane is not done in a maximally parallel but in a sequential manner (but still keeping a parallelism at the system level); the non-deterministic evolution of configuration is substituted by a definite transition following a predetermined order. This corresponds to an integer linear programming (ILP) with the subject function as a weighted sum of variables with predefined fixed weights.

In theory, membranes are borders without internal structures and material consistence. In this implementation, membrane structures are replaced by the enclosed substances, i.e., the multisets of objects, evolution rules and children membrane architectures. The objects exchange among membranes is a kind of bi-directional traversing behavior. In case of the possible objects exchange invoked in subsequent steps, the communications realized by data buses connecting to different parts of hardware are prearranged in all containment cases. The interconnections are in direct proportion to inner membranes. To avoid the multiple buses used to connect the upper-immediate membrane to its plural lower-immediate membranes, a bus links all the children membranes before it connects to the upper-immediate membrane. The communication only presents between upper-immediate membrane and lower-immediate membranes. There are no objects exchange among children membranes or non-immediate contained membranes. In general, a membrane of P system corresponding to an area of integrated circuits storing objects specifying multisets of objects and sets of rules. The containment relation of immediate-include membranes is substituted by a bus connecting them.

The representation of the multisets of objects is implemented by using registers. Different registers just preserve different multiplicities of objects. A register does not store the objects but only the vector of numbers indicating the multiplicity of each object. The order of these registers is in accordance with the lexicographic order of the alphabet of objects. The recognizing of an object is indirectly realized by examining the position of the register storing the multiplicity of this object. An evolution rule defined here is in the form of $u \rightarrow v(v_1, in_i)(v_2, out)$, where v_1 is the string to be sent into lower-immediate membrane labeled i, v_2 will be sent to upper-immediate membrane. The treatment employed to deal with the formulation of evolution rules is storing the rule's left-hand side and right-hand side into different registers separately. A particular module is designed to determine whether a rule is applicable. This module compares the left-hand side of a rule u with the multiset of objects w present in the current membrane. If and only if $u \leq w$, this rule is applicable and this module will generate a signal $Applicable = 1$. Input all the $Applicable$ signals to an OR gate, the result of this logical gate can used as a monitor to identify whether the evolution reaches halt configuration.

The transition of configurations of P system is realized deterministically and concurrently, which is different to the general model. The consecutive transformation of configurations is regarded as the evolution process. This evolution process is decomposed into micro-steps and macro-steps. The application of rules enclosed by membranes is performed in terms of a predefined sequential order. This deterministic execution of rules is conducted in micro-steps sequentially. If a selected rule is applicable, the left-hand side of the rule u will be removed. Then the right-hand sides v, v_1 and v_2 is stored in corresponding registers. The objects from the upper immediate membrane will be preserved in another register. Although the micro-steps are

carried out deterministically, they are performed simultaneously in all membranes, until there are no applicable rules. The micro-steps terminates when there are no applicable rules, i.e. the halt condition is reached. All the registers are updated in line with associated rules in macro-steps.

This implementation considered and respected the priorities of applicable rules at the beginning of each micro-step. By labeling the applicable rules with higher priorities and storing the corresponding labels, applicable rules are executed in accordance with their respective priorities. Besides, two additional features of P system, the dissolution and creation of membranes, are simulated. When a rule with membrane dissolving function is applied, its contents are owned by its upper immediate membrane, setting the membrane *Enable* signal of the relevant membrane to "0". However, the connections and registers defining the dissolved membrane still exist, for the hardware reconfiguration will cause the reconstruction of buses that connect different regions of the circuits. This scheme gives rise to a disadvantage that the hardware resources cannot be released. The creation of new membranes is executed in the initialization process of the P system since all the information about new membrane is known from the specification of the system. The created membranes are inactive until membrane creating rules invoke them.

Nguyen simulation

In this implementation, a parallel computing platform simulating membrane computing based on FPGA named Reconfig-P is developed [180, 89]. Reconfig-P is fabricated on the basis of the region-oriented idea that regions work as the computational entities communicating objects through message passing. The functionality of these regions is extended by the included set of evolution rules. P Builder, the software component of Reconfig-P, specifies the P system concerned in software, converts the specification of P system written in Java to Handel-C (a hardware description language) source codes. Software simulation of the circuits to be constructed is supported by P Builder to test the functionality of circuits before mapping the codes to hardware circuits.

The execution of a evolution step is divided into two phases: *object assignment phase* and *object production phase* [89]. The maximal instance of each rule in a region is determined in the object assignment phase. The update of multiplicity of objects is accomplished in the object production phase. The maximal instance of the rules with higher priorities is computed before the rules with lower priorities. Note that the consumption of objects for rules with higher priorities is performed during the object assignment phase to save clock cycles. It is assumed that all rules are assigned relative priorities. The priority between rules is implemented as the temporal order which should be respected by region processing units in the assignment phase. Rules with same priority are executed concurrently. The temporal order is determined at compile-time. The rules are applied according to their priorities in rounds until no rules are applicable. Under this circumstance, the applicability of each rule is non-stationary because of the existence of priorities. To avoid processing inapplicable rules, the applicability status of each rule is checked at the outset of the assignment phase and immediate after an applicable is applied to consume some objects.

The objects traversing behavior is the origin of communication between regions. The update of multiplicity of objects caused by rules with and without traversing behavior is completed in the object production phase. When different region processing units update the multiplicity value of the same object at the same time, a conflict

TABLE 3.2: The comparison of the time-oriented and space-oriented conflict resolution

Strategy	Resource conflict resolution
Time-oriented strategy	(a) construct a conflict matrix in which each row is a quadruple (p, q, r, s). p is the object competed by multiple rules. q is the region where p is produced or consumed. r is the set of the conflicting rules, s is the size of set r. (b) insert delay statement among conflict rules such that the updating operations of multiplicity of p can be executed in distinct clock cycles. The number of delays is equal to s.
Space-oriented strategy	(a) construct the conflict matrix as in the Time- oriented strategy. (b) the register storing the object accessed by multiple rules concurrently is replicated to the number of s. These copy register are assigned to each conflicting rules to write. After the updating process for all the copy registers, the corresponding values are joined to the original register.

occurs. To handle this conflict, in [89] two solution strategies, the *space-oriented strategy* and the *time-oriented strategy* are proposed. Table 3.2 and Table 3.3 summarize the different strategies of two resource conflict resolutions and their modifications in the rule-based and region-based design [162, 163]. In order to simplify the exposition of processes of rule-based and region-based implementations of P systems, tables are designed to delineate the relevant details, which will be given below. The detailed information about these two strategies is listed in Table 3.2.

For the space-oriented strategy, the register storing the object causing the conflict is replicated to the same number of parallel process rules so that the respective processing unit updates the value in the assigned copy register. For the time-oriented strategy, time delays are interleaved among the conflicting parallel processes so that the updating is performed in different time. In fact in the rule-oriented design which will be detailed hereinafter, the solution for the conflict is basically the same. The object production phase is completed in two clock cycles when the space-oriented strategy is adopted. In the time-oriented strategy, considering the traversing behavior of objects, the objects causing conflict are partitioned as *internal objects* and *external objects*. The internal objects refer to the objects generated by the rules without traversing behaviors, while the external objects refer to the opposite. The amount of interleaving inserted to the updating process caused by the internal objects is computed at compile-time. The interleaving caused by the receiving of external objects is determined at run-time for the region processing units work independently. They cannot aware of the future transferring of objects. An method involves the semaphores is adopted to determine the appropriate number of interleaving during the run-time.

The extensibility of the region-based design is the consequence of the representation of membranes as processing units interacting with two region processing units corresponding to inner and outer regions. This allows to achieve a strong separation of the processing logic inside different membranes and the independence of the communication. Thus, adding additional elements to the system does not lead to the redesign of the remaining part of the system.

3.6.2 Rule-based implementations

Rule-based approaches consider evolution rules as processing units performing the update of multiplicities and of the membrane structure.

Nguyen implementation

Every rule in all regions of the P system is represented as a processing unit synchronized by a global clock that implements the parallel processing. At the design stage, a processing unit corresponds to a potential infinite *while loop* which includes codes related to the applying of the rules in Handel-C [181, 182] code. The tags of information associated to execution and synchronization are contained in processing units as well. Each rule processing unit in a region is linked to the array of registers containing multisets of objects. The containment relationships can be described with the connections between processing units and arrays. Generally speaking, a rule processing unit in a region is linked to the objects array located in the same region with the rule processing unit. If there are objects traversing rules which imply the containment, connecting the rule processing unit to the object array to which the rule will send objects contained in different region. By this measure, the containment is reflected.

The operation of executing a rule is split into *preparation phase* and *updating phase*. In the preparation phase, each processing unit calculates the maximal number of instance for a rule with a division operation that divides the multiplicity of each objects by the number of corresponding object defined in left-hand side of a rule. A serial of quotients will obtain. Apparently, the minimum of the quotients is the maximum number of instance of a rule in terms of the Buckets Effect. In fact, the computations of maximal number of instances for rules are performed in an order with respect to the relative priorities among rules. In detail, counting the quantity of clock cycles consumed for computing the maximal instance number of rules with higher priorities at first. After that, an number of delay statements that equivalent to the number of clock cycles consumed previously are interposed above the statements for computing the maximum instance number of rules with lower priorities in the Handel-C codes. For rules with the same priority, this calculation is executed in parallel.

In the updating phase, if the maximum number of instance of a certain rule is larger than zero, then the rule is applicable. As stated before, when different processing units updates the same multiplicity value of objects registers at the same time, the conflicts arise. In order to resolve the conflict, P Builder construct a conflict matrix to detect the possibility of conflicts firstly. A row of the matrix is a four-tuple (p, q, r, s) whose p stands for an object, q is a region of the P system, r represents the set of rules gives rise to the generation or consumption of p in q, i.e. the set of rules which conflicts. s is the conflict degree of (p, q), referring to the number of conflict rules included in set r. P Builder prevents every processing unit from writing to the same register simultaneously on the basis of the analyzing the conflict matrix. Likewise, the *space-oriented strategy* and the *time-oriented strategy* are adopted to handle the conflict.

The synchronization is executed by three flags contained in the rule processing units. The region-level parallelism means that applicable rules in a region can be executed in parallel and system-level parallelism implies that performs the region-level parallelism concurrently.

To illustrate the disparity of the conflict resolution strategies used in rule-based and region-based design and these two design methodologies of Nguyen's implementation unequivocally, the contrast of two conflict resolution strategies and design methodologies are sumarized in Table 3.3 and Table 3.4.

TABLE 3.3: The differences of the conflict resolutions adopted in two design modes

Item	Time-oriented strategy	Space-oriented strategy
Rule- oriented design	The interleaving operation can be determined at compile-time and it can be hard-coded into the HDL source.	Need a multiset replication coordinator to coordinate the multiplicities stored in the copy registers.
Region-oriented design	The objects received from other regions are regarded as *external objects*, otherwise the objects are *internal objects*. The interleaving merely caused by the production of internal objects can be identified at compile-time. To retain the independence of region processing units, the interleaving induced completely or partially by the receipt of external objects can only be calculated at run-time.	The role of the multiset replication coordinator in rule-oriented design is played by the region processing units. The existing register storing the multiplicity received from the associated communication channels in the considered region can be assigned to those processing units which sending objects to the considered region to write the new values of the objects competed by multiple rules.

TABLE 3.4: The comparison of the rule-based and region-based design of Nguyen's implementation

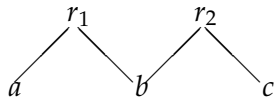
Object	Rule-based Design	Region-based Design
Region and their containment relationships	Regions are realized in hardware implicitly by the content they included. For the containment relationships including the traversing of objects between regions, they are implemented by imparting the corresponding rules with 'in' or 'out' target directives the abilities to access the multiset of objects in the destination region of the objects traversed.	Regions are represented as parallel processing units. The traversing of objects among regions is realized as message passing through channels connecting different region processing units.
Multiset of objects	An array of registers contains each type of object in the alphabet for a region	The same strategy adopted as the rule-oriented design.
Evolution rule	Potentially infinite *while loop* in which contain procedures representing the operations of the relevant evolution rules in HDL language.	Implicitly expounded through integrating them into the region processing units.
Operation process	Preparation phase and updating phase.	Object assignment phase and production phase.
Synchronization	An array of registers composed of three 1-bit registers is associated to each rule processing unit. The values in the array indicate whether the rule processing unit has complete the preparation and updating phase, or is applicable. Compute the logic AND or OR values of all the values stored in the 1-bit register that indicates the same status of the processing unit and store the three results into three sentinel registers. The coordinating processing unit read the sentinel values to synchronize the whole procedures.	For the synchronization of object assignment phase, it is realized when all the region processing units communicate with each other on channels at the beginning of object production phase. For the object production phase, a region execution coordinator connecting to each region processing unit via dedicated channels is designed to perform the synchronization of operations. After the region execution coordinator received the signals denoting the completion of all the operations of every region processing unit, the current transition is done and the next one is carried out.

Verlan and Quiros implementation

The target model is a static P system. The system is considered flattened, so only one skin membrane is present. An special strategy was elaborated in order to not compute the complete solution, i.e. $Applicable(\Pi, C, \delta)$ but the cardinality of its elements. Then a random value between 1 and this cardinality is taken. Finally, this number is decoded to the corresponding solution [90, 164].

Devising an algorithm which carries out the computation of the cardinality and of all the specific element of the solution set in constant time on FPGA is the key issue of the approach. A remarkable characteristic of FPGA is that the time consumed for executions of functions that do not exceed the cycle of the global clock is done in one cycle of FPGA, hence in constant time. The computation of the cardinality and of feasible solutions are accomplished by two functions hardwired into the circuit: $NBVariants(\Pi, C, \delta)$, which gives the cardinality of $Applicable(\Pi, C, \delta)$, and $Variant(n, \Pi, C, \delta)$, which returns the *n-th* element of $Applicable(\Pi, C, \delta)$.

A concept named rules' dependency graph is introduced to compute the two functions above. The picture below depicts the rules' dependency graph for rules $r_1 : ab \to u$ and $r_2 : bc \to v$.



Assume that the derivation mode is maximal parallelism (*max*). Suppose that N_a, N_b and N_c represent the number of objects a, b and c in C. Let $N_1 = min(N_a, N_b)$, $N_2 = min(N_b, N_c)$, $N = min(N_1, N_2)$, $k_i = N_i \ominus N$,$1 \leq i \leq 2$, where $\ominus$ denotes the positive subtraction. Let also $p, q = 0, 1, 2, \ldots, N$. From the dependency graph we can deduce the following:

$$Applicable(\Pi, C, max) = \bigcup_{p+q=N} \left\{ r_1^{p+k_1} r_2^{q+k_2} \right\}$$

$$NBVariants(\Pi, C, max) = N + 1$$

$$Variant(n, \Pi, C, max) = r_1^{N-n+1+k_1} r_2^{n-1+k_2}.$$

Example 3.1. *Consider a configuration where $N_a = 5$, $N_b = 5$ and $N_c = 3$. It can be easily verified that $N_1 = min(5,5) = 5$, $N_2 = min(5,3) = 3$, $N = min(5,3) = 3$, $k_1 = N_1 \ominus N = 5 - 3 = 2$, $k_2 = N_2 \ominus N = 3 - 3 = 0$. Hence, we can enumerate the elements of $Applicable(\Pi, C, max)$ as bellow:*

$$Applicable(\Pi, C, max)_1 = \{r_1^{3+2} r_2^{0+0}, r_1^{2+2} r_2^{1+0}, r_1^{1+2} r_2^{2+0}, r_1^{0+2} r_2^{3+0}\} = \{r_1^5, r_1^4 r_2, r_1^3 r_2^2, r_1^2 r_2^3\}.$$

The same result can be easily obtained by using formal power series associated to context-free languages. In this case, any maximal rule combination is a part of the language $L_N = \{r_1^p r_2^q \mid p + q = N\}$. It is quite easy to observe that the number of words of length N in L_N is exactly the same as the number of words of same length in the language $L = \{r_1^* r_2^*\}$. This last language is regular and its generating function is $q_0(x) = 1/(1-x)^2$. The $n - th$ coefficient of the expansion of $q_0(x)$ is equal to $n + 1$ ($[x^n]q_0 = n + 1$), which immediately gives $NBVariants(\Pi, C, max) = N + 1$. The $Variant(n, \Pi, c, max)$ is computed using an algorithm that performs a weighted breadth-first search of the decomposition of n with respect to the number of variants found on each branch of the execution of automaton for L.

Such a process can be easily repeated for any regular language, yielding a constant time simulation of a computational step. The reason for such performance is that any generating function is equivalent to a recurrence relation and such relations can be computed in one synchronous time unit using asynchronous operations. The described algorithm functions for any P system where the rule choice can be expressed as words of certain length in a regular language. The corresponding class is quite large (containing even computationally complete models), thus allowing an extremely fast execution. Examples from [90] exhibited a speed-up of order 10^5.

Another important point is that this approach allows to handle the non-determinism in a natural way, by performing a uniform random choice between all possible rule applications at each step.

Communications among different processing units do harmful to improve the computing speed. For pursuing a better speed performance, modularity is adopted to minimize the interconnections of configurable logic blocks. A layer structure which just communicates with previous one is constructed to execute the algorithm. Have in mind that P system is abstracted as a multisets rewriting system with a skin membrane, the compartment is simplified. In consequence, the rules and multisets of objects are the key materials for the implementation. As usual processing scheme for multisets of objects, registers are employed to store them in terms of configurations. While for the treatment of rules in this implementation, there is no explicit mapping from rules to hardware components, on account of the fact that not a single rule, but the dependency graph of rules is fabricated for the construction of $Applicable(\Pi, C, max)$ which can be represented as a regular language.

Technically, the implementation represents only objects by registers and rules by layered logic. Each rule implementation is modularized and contains an own copy of processing instructions needed to compute the two above functions, based on asynchronous operations. Consequently, five clock cycles are required to compute the $NBVariants(\Pi, C, max)$, $Variant(n, \Pi, C, max)$ and to apply the corresponding rules. The entire process of the implementation is split into several consecutive stages, which take charge of different operations associated to phases of evolutions of configurations.

Persistence stage stores the states that the hardware system goes through. A independent stage computes the maximal instance of each rule by means of the dividing operation and MIN logic operation. *Assignment* stage in charge of selecting a rule to be applied non-deterministically, and determines its instances. *Updating* stage is responsible for updating the current configuration with the values from the previous stage. During *Halting* stage, the system inspects whether the halting condition is reached, and once reached, stops the system.

The hardware system is separated into six blocks detailed as follows. *controlBlock* takes charge of supplying communications and control actions, including logic related to halting conditions. Given a configuration, if $Applicable(\Pi, C, max)$ is empty, or the configuration stops to evolve, then the system halts. *inoutBlock* links to the software which provides the communication with host computer. *persistenceBlock* is used for saving and updating the current configuration and partially examining the halting condition. *independentBlock*, which is independent of the derivation mode, receives the multisets of objects of the current configuration from *persistenceBlock* and carries out division and multiplication operations. The functionality of *assignBlock* corresponds to *assignment* stage. A maximum instances of an applicable rule is sent to this unit to computes $NBVariants(\Pi, C, max)$ and $Variant(n, \Pi, C, max)$. Each rule corresponds to a sub-block executing the logic of the automaton which

recognizes the regular language in terms of the dependency graph of rules. *appBlock* executes $Variant(n, \Pi, C, max)$ to modify the multiplicity of objects to evolve the configuration to next one.

The implementation of the concerned P system is achieved by the last four blocks. The four blocks consume one clock cycle to execute their work except AssignBlock, which demand two clock cycles. Consequently, five clock cycles are required to compute the $NBVariants(\Pi, C, max)$, $Variant(n, \Pi, C, max)$ and to apply the generated solution.

In order to more clearly show the FPGA implementation methods of P system proposed by the above three research groups, their methods are summarized and compared from the quantitative and qualitative perspectives in Table 3.5 and Table 3.6.

3.7 Micro-controller based implementation of P systems

Beside FPGA based hardware platforms developed for implementation of P systems, the micro-controller is another hardware device which is taken into account by the researchers. A range of P systems and their hardware circuits designs in terms of the *exhaustive investigation line* focusing on implementing the transition of configurations of P systems are developed. The nascent researches do not confine to concrete hardware devices, just designing the circuits aiming at simulating certain operations of particular P systems with registers, logical gates, magnitude comparators, and data buses.

In [183], a digital circuit is presented to select *active* rules in the current configuration. Each evolution rule is represented by 2 hardware registers. The first register characterizes the left-hand side (antecedent) of a rule, and the other specify the right-hand side of the rule, which determines whether the rewritten objects go out from the current membrane or stay where they are, or go in to inner membranes. By comparing the left-hand side of each rule with the multiset of objects in a region, the applicability of every rule can be determined. Next, an algorithm computes the number of application of active rules given the multiset of objects and evolution rules [184]. The corresponding circuit is composed of logical gates, registers, multiplexer and sequential elements. The computation process is bounded. In [185], a P system circuit is constructed by means of a micro-processor PIC16F88 plus the storage component 24LC1025, connected by an I2C bus. The shortcoming of insufficient storage capacity of micro-processor is overcame by the introduced external memory. The flexibility of the circuit is acceptable as the modification of the structure is not necessary.

As a further research of [184], in [186], the improved algorithm and its circuit calculating the application times of active rules are investigated. The computing process can be complete in minor steps and the theoretical performance is optimized. In [187, 188], a schematic implementing the inherent parallelism of P system is drafted. Towards the parallelism, the treatment to rules is similar with the thought *regional solution* defined in section 4, namely, what applied is a multiset of rules instead of a single rule. An operating environment is elaborated in [189] which performs the automatic transforming of tasks involved in the hardware simulation of P systems, including loading, execution and interpretation, into a distributed framework constructed on micro-controllers.

The execution results of the circuit, which are in the form of binary data, can be interpreted to a transparent form. What should be emphasized here is that all

TABLE 3.5: The quantitative attributes of FPGA implementation

Item	Biljana Petreska	Van Nguyen	Juan Quiros
Period	2003	2007-2010	2012-2015
Institute	Swiss Federal Institute of Technology Lausanne (EPFL)	University of South Australia	University of Seville
Target FPGA	Xilinx Virtex-II Pro 2VP50ff1517-7	Xilinx Virtex-II XC2V6000-ff1152-4 (rule-oriented) and Virtex-II RC2000 (region-oriented)	Xilinx Virtex-V XC5VFX70T and Virtex-VII XC7VX485T
Host processing platform	Not given.	1.73GHz Intel Pentium M processor with 2GB of memory	Intel Core i5 – 5220 at 3 GHz, with 8 GB of RAM.
HDL	VHDL	Handle C	VHDL
Experiment Subjects	Cell-like P systems with following characteristics: membrane dissolution and creation, objects exchange between upper- and lower-immediate membranes, cooperative P systems with priorities. The range (non-continuous) for object number is 6 to 12 and for membrane is 10 to 20.	For rule-oriented design, the subjects are cell-like P systems cascaded in vertical, horizontal, vertical and horizontal structures. For region-oriented design, the objects are cell-like P systems containing hierarchical regions and tissue-like connected regions. The rule range is [10,50], [1,25] for regions, [3,200] for objects. The extent of the inter-region communication is [80,319].	The subject P system is simplified according to the multiset rewriting point of view, which it has a skin membrane, no inner regions. The 4 subjects differ in the rule dependencies which form chains: circular, 2-circular, linear, opposite. The object number range is [10,200].
Experiment results	The hardware consumption ranges from 4.2% to 33% (CLB). The extent of clock rate is 27 to 198 MHz.	For rule-oriented design, the number of rules applied per second ranges from 2.7×10^5 to 1000×10^5. The hardware consumption extent is 1.55% to 21.43% (LTU). For region-oriented design, the hardware usage ranges 1.82% from 16.79%. The clock rate fluctuates from 52.63 MHz to 81.77 MHz. The biggest size which can be executed is a P system with 550 rules, 1280 communication channels, 1100 objects conflicts.	The hardware consumption ranges from nearly 1% to 48% (LUT) or nearly 1.1% to 11% (slice). The period needed to perform a computation step fluctuates from 5.46 *ns* to 9.14 *ns*. The highest frequency exceeds 100 MHz, permitting 2×10^7 computational steps per second. The runtime for each experiment subject ranges from $3.017 \times 10^{-5}s$ to $4.174 \times 10^{-5}s$.
Parallelism	Region-level parallelism	Region-level and system-level parallelism	System-level (there is only a skin membrane, so it is also region-level parallelism).
Non-determinism	No non-determinism	DND algorithm	True non-determinism.

TABLE 3.6: The qualitative attributes of FPGA implementations

Item	Biljana Petreska	Van Nguyen	Juan Quiros
Membranes (regions) and their containment	Implicitly represented by the contents enclosed by the membranes. The objects exchange is interpreted as transferring objects with communication buses by which connect the origin region to the destination region.	For the rule-oriented design, the treatment is similar with Petreska's. For the region-oriented design, regions are represented as parallel processing units. The objects exchange among regions is realized as message passing through channels connecting different region processing units.	According to the multiset rewriting system framework, the topology structures of membranes are not important. What concerned is the rule dependency graph.
Multiset of objects	Store the multiplicity value of each type of object in different registers whose positions indicate the type.	The same method.	The same method.
Evolution rule	Store the left-hand-side and right-hand-side of a rule in different registers.	In rule-oriented design, they are characterized as potentially infinite *while loop* in which contain procedures representing the operations of the relevant evolution rules in HDL. In region-oriented design, they are implicitly expounded through integrating them into the region processing units.	The logic of rules is distributed along the the hardware components. There is no explicitly correspondence between rules and hardware components.
Operation Process	Micro-step: only one applicable rule is applied with respect to the instance number in a region. Macro-step: execute a micro-step concurrently in every region.	In rule-oriented design, perform preparation phase and updating phase. In region-oriented design,perform object assignment phase and production phase.	1. Persistence stage 2. Independent stage 3. Assignment stage 4. Application stage 5. Updating stage 6. Halting stage.
Extensibility	Membrane mediated features cannot be added in since membranes are implicitly represented by their contents.	Because of the region-oriented design, Membrane mediated features such as *symport* and *antiport* functions can be extended.	This implementation is designed only for P systems whose applicable multisets of rules can be represented as regular context-free language. So its extensibility is limited.
Scalability	With the increase of the rules, the hardware usage rise approximately proportional. The clock rates decline a small amount. The membrane creation ability cripples clock rate significantly.	The hardware consumption scales linearly with respect to the size of the P system executed in both rule-oriented and region-oriented design. The performances grow linearly when the number of rules increase.	The hardware usages are scalable, but the rate of increase in the performance is not always linear for different subjects with distinct structures.
Contributions	Membrane creation and dissolution functionality	Two kinds of methodology for P system characterization and resource conflict resolution, DND algorithm	Put up with a new methodology with absolute equiprobability to implement non-determinism.
Drawback	Partially parallelism, no non-determinism	The equiprobability of DND algorithm has not been proven theoretically.	The types of P systems which can be implemented are confined to those whose rules can be represented as regular context-free language.

the hardware circuits introduced are just on the blueprints which do not put into practice. They are theoretical analysis, the actual functionality and performances are unknown, unlike the FPGA based and CUDA based hardware implementations/simulations which are carried out practically. In [190], micro controllers are also chosen as target hardware to implement communication architectures of P systems. A digital circuit carrying out massive parallelism in transition P Systems is established in [191].

As a new attempt for implementing neural P systems on different hardware, DRAM-based CMOS circuits are adopted to construct elementary Spiking neural P systems, which had not been implemented on FPGA but on CUDA hardware. We do not carry out an in-depth discussion about this topic given the length of the article, interested people can refer to [192] for more details. In [193], A preliminary microfluidic system, called μfluidic P system, is developed for a spiking neural P system aiming at solving the Boolean satisfiability SAT problem. An adder elaborated from a spiking neural P system is proposed in [194]. This adder which has good error tolerance for the using of dual-rail logic can perform addition of binary numbers with arbitrary length. These are the newly achieved progresses of hardware implementation of P systems.

3.8 Wrap-up

The internal fabric of SRAM-LUT based FPGA is presented squarely to decipher the origins of reconfigurability and parallelizability. FPGA developing flow is rendered to acquaint reviews of FPGA implementation and works done in ensuing chapters. No hardware implementations of RS are found in literature for the nonce so software simulations are introduced instead. The CUDA GPU architecture is introduced to reveal its working pattern. FPGA implementations of P systems are combed and dissected so that the research lines, difficulties, existing and potential methods are expounded completely. Implementing the non-determinism of P systems on hardware is a intractable problem to be quested with more intelligences. Micro-controller based implementation of P systems is inquired and reviewed, although their performances are far behind FPGAs'.

Chapter 4

FPGA implementation of reaction systems

Reaction systems (RSs) are parallel models in which applicable reactions execute concurrently. For large scale RSs containing thousands of reactions, the efficiency of CPU based simulation is low. GPU accelerated simulation shows speed-up of 26.28× on Python based and 10.97× on Haskell based simulation [145]. FPGA based implementation would achieve a better performance for parallel hardware architectures tailored to coincide with RS models to carry out reactions with a dizzying speed. This chapter proposes FPGA implementation approach for two RS models: self assembly of intermediate filaments RS and heat shock response (HSR) RS. A special binary counter RS which is a quantitative model constructed by qualitative one is implemented as well to mirror the expressivity of RS and the vague demarcation between qualitativeness and quantitativeness.

4.1 Relations between reaction systems and synchronous circuits

Digital systems consist of synchronous circuits which can be defined as Definition 4.1 belonging to a type of ideal model. Physical circuits cannot hold these two properties since they are too rigid. In engineering field, synchronous circuits refer to circuits which are approximate to the perfect one defined by the definition.

Definition 4.1. [195] *A circuit is said to be synchronous if it possesses the following two characters:*

1. *Any lead or device within the circuit may assume, at any instant of time, only one of two conditions, such as high or low voltage, pulse or no pulse.*
2. *The behavior of the circuit may be completely described by the consideration of conditions in the circuit at equally-spaced instants in time.*

To judge whether a circuit is synchronous or not, the following criterion containing three items which should be satisfied simultaneously is commonly accepted [195].

1. A clock working as trigger is present.
2. Current or voltage signals used as input/output variables arise concurrently with clock trigger edge.
3. If inputs are synchronous with clock trigger edge, clock period can vary some extend and output right results.

The switching circuit [196, 197] defined in Definition 4.2 is the theoretical model of physical synchronous circuits fabricated in electronic devices such as a FPGA. The state allows us to memorize past values and to perform decisions based on the partial history of the computation. Hence, in this case the value of the function may be different for same inputs at different time steps (usually corresponding to the master clock pulses that drive the circuit).

The functioning of a sequential switching circuit with n inputs and m outputs and s binary-state variables can be described by the following equations [198]:

$$\begin{aligned} Q(t+1) &= F(Q(t), X(t)) \\ Y(t) &= G(Q(t), X(t)), \end{aligned} \tag{4.1}$$

where $X(t) = (x_1(t), \ldots, x_n(t))$ is the vector of input variables at time $t \geq 0$, $Y(t) = (y_1(t), \ldots, y_m(t))$ is the vector of output variables at time t, $Q(t) = (q_1(t), \ldots, q_s(t))$ is the vector of internal states at time t, $F : \{0,1\}^s \times \{0,1\}^n \to \{0,1\}^s$ and $G : \{0,1\}^s \times \{0,1\}^n \to \{0,1\}^m$.

Definition 4.2. [195] *The switching circuit is a synchronous circuit with finite numbers of inputs, outputs, and (internal) states. Its present output combination and next state are determined uniquely by the present input combination and the present state. A circuit having only one internal state is termed as combinatorial circuit otherwise it is referred to as sequential circuit.*

Behaviors of sequential circuits are described by their corresponding truth tables, which can be transformed to state diagrams that demonstrate the dynamics explicitly. Mealy and Moore automata [199] are proposed based on these state diagrams. Their definitions are given as follows.

Definition 4.3. [195] *Mealy automaton is an finite state machine whose output depends on the current state together with current input. It is a 6-tuple (Q, Σ, O, X, Y, q_0) where*

1. *Q is a finite set of states;*
2. *Σ denotes the input alphabet;*
3. *O is output alphabet;*
4. *X is the input transition function mapping $Q \times \Sigma \to Q$;*
5. *Y is the output transition function mapping $Q \times \Sigma \to O$;*
6. *q_0 is the initial state ($q_0 \in Q$).*

Definition 4.4. [200] *Moore automaton is an finite state machine whose output depends on the current state solely. It is a 6-tuple (Q, Σ, O, X, Y, q_0) where*

1. *Q is a finite set of states;*
2. *Σ denotes the input alphabet;*
3. *O is output alphabet;*
4. *X is the input transition function mapping $Q \times \Sigma \to Q$;*
5. *Y is the output transition function mapping $Q \to O$;*
6. *q_0 is the initial state ($q_0 \in Q$).*

TABLE 4.1: The truth table of Formula 4.2.

$q_1(t)$	$q_2(t)$	$x_1(t)$	$q_1(t+1)$	$q_2(t+1)$	$y_1(t)$
0	0	0	1	0	1
0	0	1	1	0	1
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	1	1	0
1	0	1	0	1	0
1	1	0	0	0	0
1	1	1	0	1	0

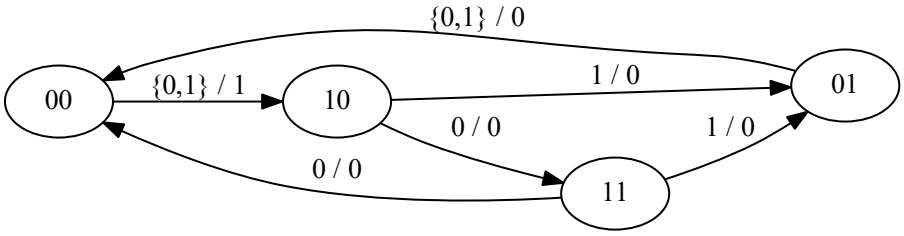


FIGURE 4.1: The Mealy automaton for the circuit described by Equation (4.2). The label of the state corresponds to the value of the vector (q_1, q_2). The label of the transition corresponds to the value of the input variable x and output y.

A sequential circuit presented in [195] is described by Equation (4.2). Its truth table is given as Table 4.1. The dynamics of state transition of the circuit can be characterized by the Mealy automaton illustrated in Figure 4.1.

$$\begin{aligned} q_1(t+1) &= \bar{q}_1(t) \wedge \bar{q}_2(t) \vee \bar{x}_1(t) \wedge \bar{q}_2(t) \\ q_2(t+1) &= q_1(t) \wedge \bar{q}_2(t) \vee x_1(t) \wedge q_1(t) \\ y_1(t) &= \bar{q}_1(t) \wedge \bar{q}_2(t). \end{aligned} \tag{4.2}$$

The corresponding Moore automaton of Equation (4.2) is depicted in Figure 4.2.

A sequential circuit can be transformed to a Mealy or Moore automaton for their are equivalent although Moore machine has a one step delay. Reaction systems are implemented as switching circuits for their relationship detailed below.

4.1.1 From reaction systems to switching circuits

The definition of reaction systems is presented in Chapter 2 Section 1. Now the normal form of reaction system is given bellow.

Definition 4.5. *A reaction system $\mathcal{R} = (S, A)$ is said to be in a normal form with respect to the interactive process $\pi = (\gamma, \delta)$ if*

- *for any $a = (R, I, P) \in A$ it holds $|P| = 1$ (i.e., only one product is allowed per reaction),*
- $\bigcup_{(R,I,P)\in A} P \cap \bigcup_{i\geq 0} C_i = \emptyset$ *(i.e. the set of products is disjoint with the set of contexts).*

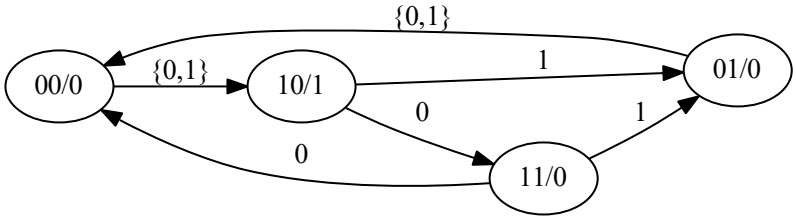


FIGURE 4.2: The Moore automaton for the circuit described by Equation (4.2). The label of the state corresponds to the value of the vector $(q_1, q_2)/y$. The label of the transition corresponds to the value of the input variable x.

Define the input for a reaction system $inp(\mathcal{R})$ as the set of all possible context symbols: $inp(\mathcal{R}) = \{Z \in \delta \mid \text{ for any interactive process } \pi = \{\gamma, \delta\}\}$. Then define the output of a RS as a projection of background set S: $out(\mathcal{R}) \subseteq S$. Now construct the Equation (4.1) using the method from [25], that transforms a reaction system to a Boolean formula [201] in disjunctive normal form (DNF).

Suppose that a reaction system $\mathcal{R} = (S, A)$ in a normal form with input $\mathcal{I}$. Then each group of reactions having the same product $A_p = \{(R_a, I_a, p) \mid a \in A\}$ can be seen as the following equation:

$$p(t+1) = \bigvee_{(R_a, I_a, p) \in A_p} \left(\bigwedge_{X \in R_a} X(t) \wedge \bigwedge_{Y \in I_a} \bar{Y}(t) \right). \tag{4.3}$$

In order to compute the output of the switching circuit, add the following equation (since it is a projection, we may omit it if it is clear from the context):

$$y(t) = y(t), \quad \text{for all } y \in out(\mathcal{R}). \tag{4.4}$$

Equations (4.3) and (4.4) are of the form of Equation (4.1), so they define a switching circuit. Moreover, since sets of reactants and inhibitors are disjoint, formula (4.3) is in DNF.

Example 4.6. *In [36], a model for the self-assembly of intermediate filaments from vimentin tetramers is presented. The first model from that paper is considered here (the other more complex variants of the model is implemented in FPGA in subsequent section). It is defined as follows.*

The background set is $S = \{O, H, F, d\}$ and the input set is $\{T\}$. The reactions are the following (d is the dummy inhibitor):

$$\begin{aligned}
&(\{T\}, \{d\}, \{O\}),\\
&(\{O\}, \{d\}, \{H\}),\\
&(\{H\}, \{d\}, \{F\}),\\
&(\{F\}, \{d\}, \{F\}).
\end{aligned}$$

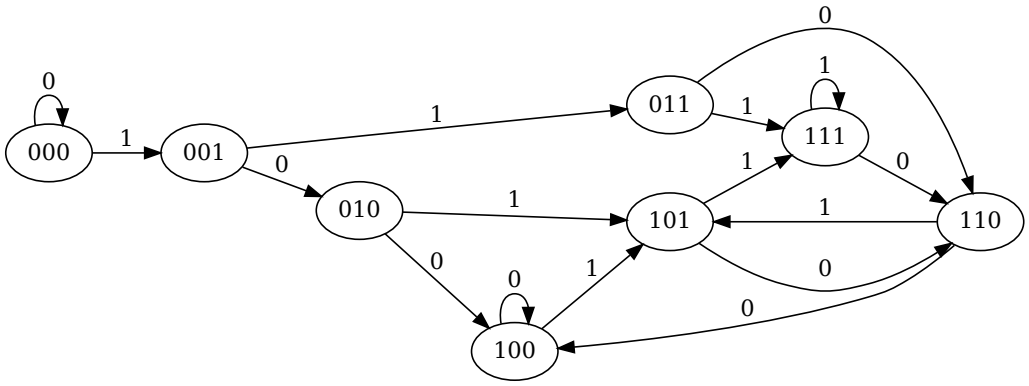


FIGURE 4.3: Moore machine for Example 4.6. The state label corresponds to the values of the vector (F, H, O) and the transitions are labeled by the value of T. The output is the label of the state.

Using Equations (4.3) *and* (4.4) *we obtain the following sequential circuit:*

$$\begin{aligned} d(t+1) &= 0 \\ O(t+1) &= T(t) \wedge \bar{d}(t) \\ H(t+1) &= O(t) \wedge \bar{d}(t) \\ F(t+1) &= (H(t) \wedge \bar{d}(t)) \vee (F(t) \wedge \bar{d}(t)). \end{aligned}$$

The Moore machine for this circuit is depicted on Figure 4.3. It can be immediately deduced that there is a steady loop between states 101, 110 *and* 111 *corresponding to the last rule that keeps F indefinitely once produced.*

4.1.2 From switching circuits to reaction systems

Let C be a switching circuit with n inputs, m outputs and s internal states described by Equation (4.1). So C can be portrayed by Equation (4.5) in which q_i and y_i are one element of Q and Y in Equation (4.1). Without loss of generality, suppose that F and G are in disjunctive normal form.

$$\begin{cases} q_i(t+1) = F_i(Q(t), X(t)) \\ y_j(t) = G_j(Q(t), X(t)) \end{cases} \tag{4.5}$$

The function of switching circuits can be described by Boolean expressions, which can be written in disjunctive normal forms. So

$$q_i(t+1) = F_i(Q(t), X(t)) = \bigvee_{1 \le s \le e} c_s$$

where c_s is the conjunctions of $Q(t)$ and $X(t)$. Similarly, $y_j(t) = G_j(Q(t), X(t))$ can be transformed to a certain disjunctive normal form over conjunctions of $Q(t)$ and $X(t)$.

Then construct a reaction system $\mathcal{R}$ with input $inp(\mathcal{R}) = (S, I)$, where $S = Q \cup Y$ and $I = X$. The reaction set A is defined as follows:

Let a be a conjunction $a = a_1 \wedge \cdots \wedge a_{k_1} \cdots \wedge a_{k_2}$, $0 < k_1 < k_2$, where $a_1 \sim a_{k_1}$ are reactants, $a_{k_1+1} \sim a_{k_2}$ are inhibitors, and define

$pos(a) = \{a_s \mid a_s \text{ is a positive literal, } 1 \le s \le k_1\}$

$neg(a) = \{a_s \mid a_s \text{ is a negative literal, } k_1 < s \le k_2\}$.

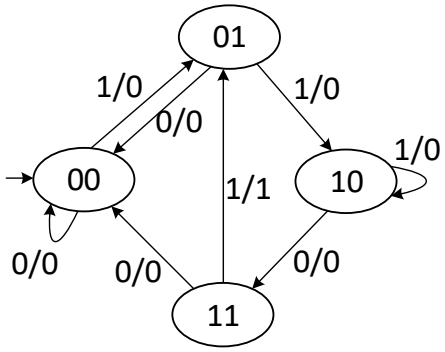


FIGURE 4.4: Mealy machine for the 1101 sequence detector. It outputs 1 when the corresponding sequence is encountered as input. The state label corresponds to the vector (q_2, q_1).

Positive literals denote reactants which are not negated while negative literals denote inhibitors which are negated (signifying the fact that inhibitors should be absent). As a consequence, $q_i(t+1) = F_i(Q(t), X(t)) = \bigvee_{1 \leq s \leq e} c_s$ can be converted to the serials of reactions in Equation (4.6):

$$\begin{cases} (\{pos(c_1)\}, \{neg(c_1)\}, \{q_i\}) \\ (\{pos(c_2)\}, \{neg(c_2)\}, \{q_i\}) \\ \vdots \\ (\{pos(c_e)\}, \{neg(c_e)\}, \{q_i\}) \end{cases} \tag{4.6}$$

In Equation (4.6), $\{q_i\}$ is the product of multiple reactions which are connected with $\vee$ to form a disjunctive normal form. In the same way, $y_j(t) = G_j(Q(t), X(t))$ can be transformed to some reactions of $\mathcal{R}$. Now, the initial values of state variables of the circuit give the value C_0 of the initial context for any interactive process π for this RS. So it is concluded that a switching circuit can be transformed to a reaction system.

Example 4.7. *Consider the circuit that implements a sequence detector and outputs 1 if the sequence* 1101 *is detected as input. The corresponding Mealy machine is depicted in Figure 4.4 and the corresponding truth table is given in Table 4.2.*

From this table it can be deduced the state equations of the circuit (x being the input bit and y the output result):

$$\begin{aligned} q_2(t+1) &= \bar{q}_2(t) \wedge q_1(t) \wedge x \vee q_2(t) \wedge \bar{q}_1(t) \\ q_1(t+1) &= \bar{q}_2(t) \wedge \bar{q}_1(t) \wedge x \vee q_2(t) \wedge \bar{q}_1(t) \wedge \bar{x} \vee q_2(t) \wedge q_1(t) \wedge x(t) \\ y(t) &= q_2(t) \wedge q_1(t) \wedge x(t) \end{aligned}$$

TABLE 4.2: The truth table for Example 4.7.

$q_2(t)$	$q_1(t)$	$x(t)$	$q_2(t+1)$	$q_1(t+1)$	$y(t)$
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	1	1	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	1	0	1	1

Using the above algorithm these equations are transformed to the following reaction system (where d is the dummy inhibitor and initially the system is empty):

$$\{x\}, \{q_1, q_2\}, \{q_1\},$$
$$\{q_2\}, \{q_1, x\}, \{q_1\},$$
$$\{q_1, q_2, x\}, \{d\}, \{q_1\},$$
$$\{q_1, x\}, \{q_2\}, \{q_2\},$$
$$\{q_2\}, \{q_1\}, \{q_2\},$$
$$\{q_1, q_2, x\}, \{d\}, \{y\}.$$

4.1.3 General ideas for FPGA implementation of reaction systems

From the discussion above, a RS can be transformed to sequential circuits. As the consequence, FPGA implementation of RS is the process to find out the equations of the circuits at first, then to design circuits to perform these equations according to the dynamics of RS. To this end, several questions should be considered.

The first consideration is how to characterize a RS in FPGA, namely, how to represent entities, reactions, and interactive process. As a qualitative framework, the RS works on sets instead of multisets. The amount of a entity (multiplicity) makes no sense taking into account the *threshold supply* assumption: if an entity presents, it is sufficient so that reactions containing it as a reactant can take place simultaneously; or it is absent. The "+" symbol in the both sides of a reaction means that more than one type of reactants should present together to produce multiple products. The interactive process is powered by the introducing of context sequence C_i for entities fade away because of *no permanency* assumption. These three issues are attacked as follows.

Entities representation. Entities are represented by 1-bit registers such that logic 1 denotes present while logic 0 indicates absent. By this way, the non-quantitative attribute of *threshold supply* assumption is reflected. Reactions involve multiple reactants are applicable only if values in relevant registers are 1, otherwise they are inapplicable.

Reactions expression. For reactions having multiple reactants, logic AND is employed to denote "+" symbol to represent all reactants should be present at the same

time. Registers storing inhibitors are inverted by logic NOT to signify that they are not here. Note that reactants and inverted inhibitors are connected with AND to represent the fact that reactions can proceed with the presence of all reactants and without all inhibitors. In one word, reactions are translated into logic expressions.

Dynamics of interactive process. *No permanency* assumption stipulates the duration of a entity is one step of the interactive process. The time interval between C_i and C_{i+1} is one step duration which is the default time bound [32, 34]. Duration is interpreted as FPGA clock period. One entity persists more than one duration if it is produced by some reactions or introduced in as an element of some C_i. Or else the maximal duration is one clock cycle.

Entities in reactions are invariant while C_i may bring in new entities during the process. These new entities are taken into consideration by assigning 1-bit registers to them when check all C_i at the system specification from the very beginning. A register variable named counter is used to count the clock cycles corresponding to interactive steps. Consequently, the time an entity led in is denoted by the integer value saved in the counter register. The dynamic process of a RS is portrayed by a waveform consisting of entities logic value varying with clock cycles.

Hardware facilities used in RS implementations of this chapter include: a Dell *Latitude* equipped with a Intel Core i7-7820HQ and 16 GB RAM is the host computer. The FPGA developing board is Digilent BASYS 3 featured with a Xilinx Artix-7 xc7a35tcpg236c-1 FPGA. FPGA integrated developing environment (IDE) is Xilinx Vivado 2019.1 which is a new generation software dedicated to developing 7 series and UltraScale FPGAs.

4.2 FPGA implementation of self-assembly intermediate filaments reaction systems

The cytoskeleton [202] of eukaryotic cells constitutes three protein filaments: microtubules, actin filaments and intermediate filaments (IFs) [203, 204]. In the (*in-vitro*) self-assembly process of intermediate filaments, monomers polymerize to tetramers (signified by T) through dimers. While this stage is usually skipped so tetramers is the initial objects. In the next stage, double tetramers concatenate laterally to one octamer (O) then two octamers combine together to a hexadecamer (H). At last a couple of hexadecamers connect to one unit length filament (F_u). An elongation reaction associates two F_us to generate a *short filaments* F_s longer than F_u. With merger reactions, F_ss and/or F_us are attached to form *long filaments* F_ls (containing at least 3 F_us). Filaments composed of F_l are compressed radially in the last stage which is omitted too since the components do not change [205]. The corresponding molecular reactions and RS counterparts of IFs presented in [36] is depicted in Table 4.3.

Accordingly, the RS of IF is $\mathcal{R}_{ifl1} = (B_1, A_1)$ where B={T,O,H,F_u,F_s,F_l,d_i}, A comprises the reactions listed in Table 4.3. The interactive process having 6 steps (initial step 0 is excluded) is created by feeding T as the only entity of the context sequence, given in Table 4.4 [36].

As stated in the general ideas of FPGA implementation of RS, entities in applicable reactions in each interactive process are represented as logic 0 or 1 depending on their absences and presences. This technique leads to that the interactive process is manifested with waveforms of entities involved. The waveform of IF interactive process corresponding to Table 4.4 is portrayed in Figure 4.5 (a). Design the RTL model of $\mathcal{R}_{ifl1}$ in the light of general ideas about FPGA implementation of RS and

TABLE 4.3: Molecular reactions and their RS equivalents of self-assembly intermediate filaments.

No.	Molecular reactions	RS reactions
(1)	$2T \rightarrow O$	({T},{d_i},{O})
(2)	$2O \rightarrow H$	({O},{d_i},{H})
(3)	$2H \rightarrow F_u$	({H},{d_i},{F_u})
(4)	$2F_u \rightarrow F_s$	({F_u},{d_i},{F_s})
(5)	$F_u + F_s \rightarrow F_l$	({F_u,F_s},{d_i},{F_l})
(6)	$F_u + F_l \rightarrow F_l$	({F_u,F_l},{d_i},{F_l})
(7)	$2F_s \rightarrow F_l$	({F_s},{d_i},{F_l})
(8)	$F_s + F_l \rightarrow F_l$	({F_s,F_l},{d_i},{F_l})
(9)	$2F_l \rightarrow F_l$	({F_l},{d_i},{F_l})

TABLE 4.4: Interactive process of $\mathcal{R}_{ifl1} = (B_1, A_1)$ (6 steps) with T as the sole entity of each context for intermediate filaments RS.

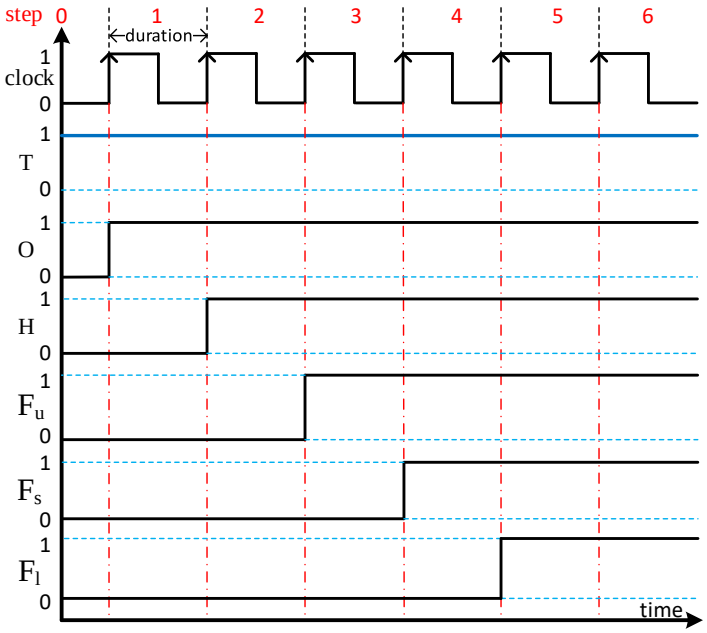
Step	C_i	D_i	W_i	Applicable reactions
0	{T}	∅	{T}	(1)
1	{T}	{O}	{T,O}	(1),(2)
2	{T}	{O,H}	{T,O,H}	(1)∼(3)
3	{T}	{O,H,F_u}	{T,O,H,F_u}	(1)∼(4)
4	{T}	{O,H,F_u,F_s}	{T,O,H,F_u,F_s}	(1)∼(6)
5	{T}	{O,H,F_u,F_s,F_l}	{T,O,H,F_u,F_s,F_l}	(1)∼(9)
6	{T}	{O,H,F_u,F_s,F_l}	{T,O,H,F_u,F_s,F_l}	(1)∼(9)

perform the behavioral simulation to estimate the functionality of the RTL model. The behavioral simulation waveform of $\mathcal{R}_{ifl1}$ RTL model generated by Vivado is given in Figure 4.5 (b). From the contrast of these two waveforms, the designed RTL model of $\mathcal{R}_{ifl1}$ fits in with expectation of its function, so the RTL model is correct.

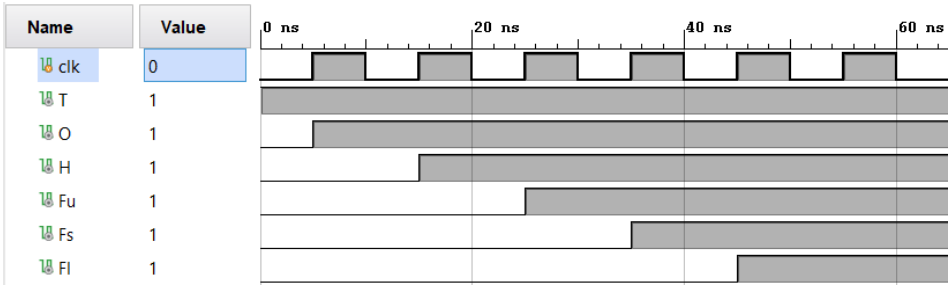
The length of produced filaments can be controlled by adding inhibitors to some reactions, adding new entities in the context or getting rid of some reactions in Table 4.4. A more complex variant, $\mathcal{R}_{ifl2}$=(B_2, A_2) where $B_2 = \{T, O, H, F_u, F_s, F_l, long, short, F_{long}, d_i\}$ is devised in [36] to generate filaments whose lengths varies with context entities. New inhibitors are appended to several reactions to substitute dummy inhibitor d_i and a new reaction is introduced. Reaction set A_2 is listed in Table 4.5. The interactive process is given in Table 4.6. The expected waveform of Table 4.6 shown in Figure 4.6 (a) is verified by the behavioral simulation waveform of its RTL model given in Figure 4.6 (b).

TABLE 4.5: RS reactions of $\mathcal{R}_{ifl2} = (B_2, A_2)$.

No.	RS reactions
(1)	({T},{d_i},{O})
(2)	({O},{d_i},{H})
(3)	({H},{d_i},{F_u})
(10)	({F_u},{F_{long}},{F_s})
(11)	({F_u,F_s},{short},{F_l})
(12)	({F_s},{short},{F_l})
(13)	({F_u,F_l},{short},{F_l})
(14)	({F_s,F_l},{short},{F_l})
(15)	({F_l},{short},{F_l})
(16)	({F_l,long},{d_i},{F_{long}})



(a) The expected waveform of interactive process listed in Table 4.4. The blue solid line indicates context T.

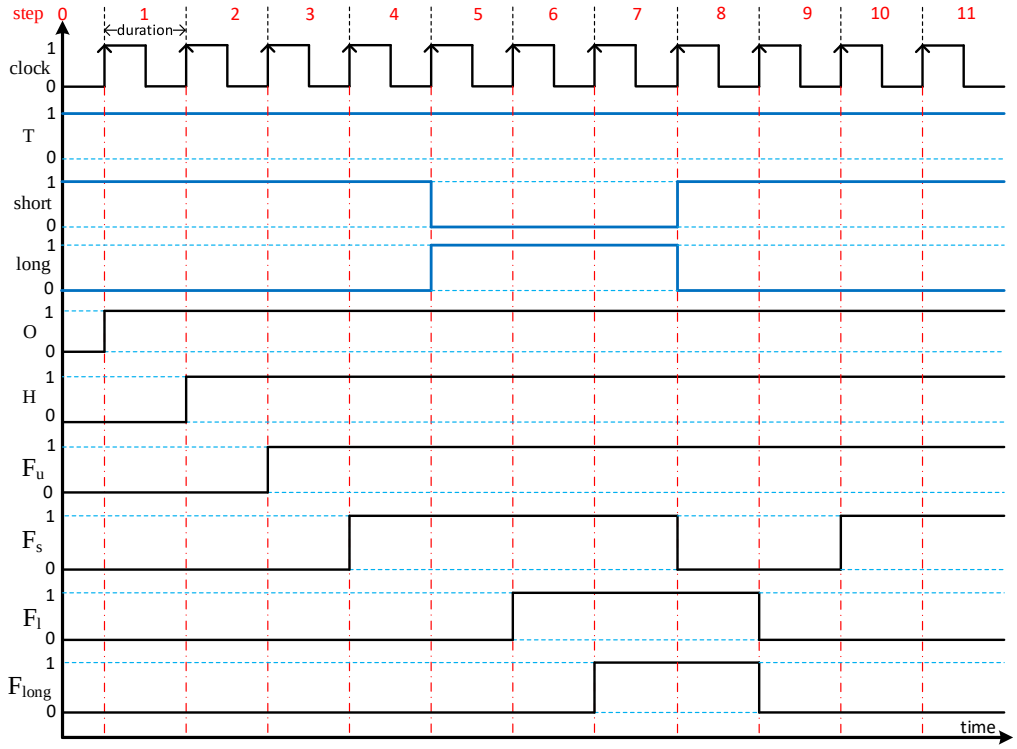


(b) Vivado behavioral simulation waveform of interactive process of $\mathcal{R}_{ifl1}$. The clock perid is set as 10 ns.

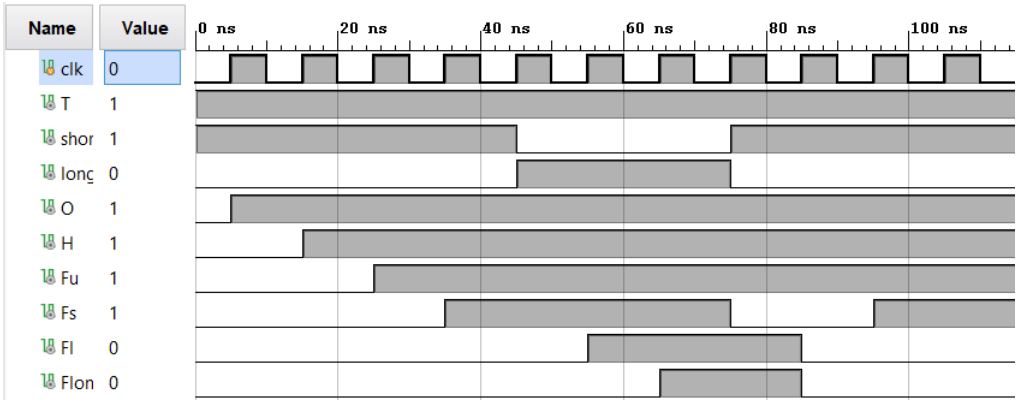
FIGURE 4.5: Waveform comparison of interactive process of $\mathcal{R}_{ifl1} = (B_1, A_1)$.

Since the RTL models behave exactly as expected behaviors, it is ready to proceed the FPGA development flow. After synthesizing these RS models, set physical constraints to establish the mapping between RTL model ports to FPGA package pins, called I/O planning as shown in Figure 4.7. Set the clock period of the two models as 5 ns as the timing constraint. The output ports of the two RTL models are marked as debug signals so that their waveforms can be observed in the hardware debug window after programming FPGA with design bitstream file. Then conduct implementation which is performed by Vivado automatically. The timing report generated after implementation tells whether timing requirements are met or not. In fact, 4 ns clock period failed to meet timing requirements so it is increased to 5 ns which meets all timing requirements.

Post implementation timing simulation is a significant procedure to evaluate the performance of a designed model because gate delays and datapath delays are considered. During simulations, the clock period is defined by setting the half period and real numbers are rounded to nearest integers. This means if the clock period is an odd number, the actual clock is 1 ns longer for the 0.5 is rounded to 1. This is the



(a) The expected waveform of interactive process listed in Table 4.6. The blue solid lines indicate context entities.



(b) Vivado behavioral simulation waveform of interactive process of $\mathcal{R}_{ifl2}$. The clock perid is set as 10 ns.

FIGURE 4.6: Waveform comparison of interactive process of $\mathcal{R}_{ifl2} = (B_2, A_2)$.

TABLE 4.6: Interactive process (11 steps) of $\mathcal{R}_{ifl2} = (B_2, A_2)$.

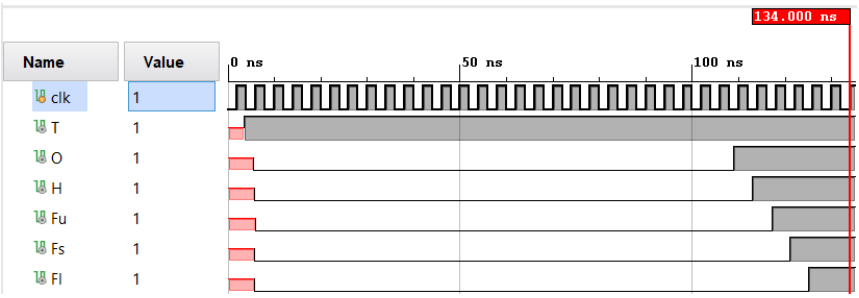
Step	C_i	D_i	W_i	Applicable reactions
0	{T,short}	$\emptyset$	{T,short}	(1)
1	{T,short}	{O}	{T,O,short}	(1),(2)
2	{T,short}	{O,H}	{T,O,H,short}	(1)∼(3)
3	{T,short}	{O,H,F_u}	{T,O,H,F_u,short}	(1)∼(3),(10)
4	{T,short}	{O,H,F_u,F_s}	{T,O,H,F_u,F_s,short}	(1)∼(3),(10)
5	{T,long}	{O,H,F_u,F_s}	{T,O,H,F_u,F_s,long}	(1)∼(3),(10)∼(12)
6	{T,long}	{O,H,F_u,F_s,F_l}	{T,O,H,F_u,F_s,F_l,long}	(1)∼(3),(10)∼(16)
7	{T,long}	{O,H,F_u,F_s,F_l,F_{long}}	{T,O,H,F_u,F_s,F_l,F_{long},long}	(1)∼(3),(11)∼(16)
8	{T,short}	{O,H,F_u,F_l,F_{long}}	{T,O,H,F_u,F_l,F_{long},short}	(1)∼(3)
9	{T,short}	{O,H,F_u}	{T,O,H,F_u,short}	(1)∼(3),(10)
10	{T,short}	{O,H,F_u,F_s}	{T,O,H,F_u,F_s,short}	(1)∼(3),(10)
11	{T,short}	{O,H,F_u,F_s}	{T,O,H,F_u,F_s,short}	(1)∼(3),(10)

(a) I/O planning of $\mathcal{R}_{ifl1}$. (b) I/O planning of $\mathcal{R}_{ifl2}$.

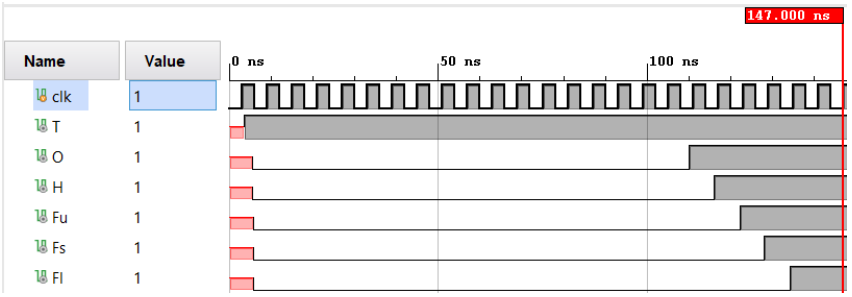
FIGURE 4.7: Physical constraints settings of two intermediate filaments reaction systems. Gray circles with wide orange bars are pins allocated to RTL model ports.

case for the implementation of two intermediate filaments RSs because the clock period is 5 ns. To assess the performance of two models, two post implementation timing simulations with 4 ns and 6 ns as clock period are carried out, shown in Figure 4.8 and Figure A.1 in the Appendix. The mean integer value, 141 ns and 166 ns are selected as the elapsed time for 6 steps of $\mathcal{R}_{ifl1}$ and 11 steps of $\mathcal{R}_{ifl2}$ model. 5 ns clock period implies that the computation speed is 2×10^8 step per second and its corresponding frequency is 200 MHz. Computing logic expressions composed of 1-bit variables is very efficient comparing to arithmetic expressions involving multiple-bit variables. This is the root cause of the high performance of FPGA implemented RSs.

After post implementation simulation completed, perform generating bitstream file and programming device with this file. As output port variables are marked as debug signals in the Verilog codes, the integrated logic analyzer (ILA) dashboard appears after device programming. Add all the marked signals to Trigger/Capture setup windows and run, results calculated by FPGA are shown in the dashboard in



(a) Post implementation timing simulation with 4 ns clock period.



(b) Post implementation timing simulation with 6 ns clock period.

FIGURE 4.8: Post implementation timing simulation of $\mathcal{R}_{ifl1}$.

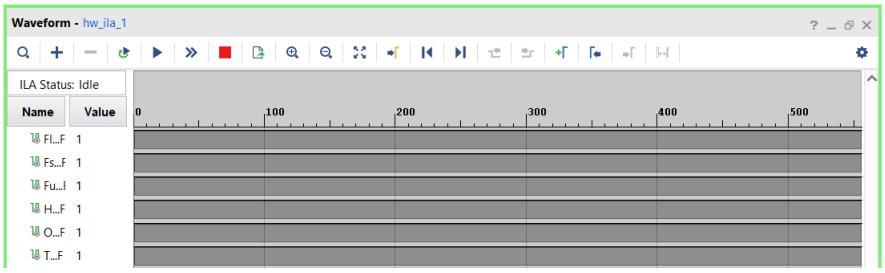
the form of waveforms, as can be seen in Figure 4.9. Hardware resources dissipation and power consumption is given in Figure 4.10 and A.2 in the Appendix.

4.3 FPGA implementation of heat shock response reaction systems

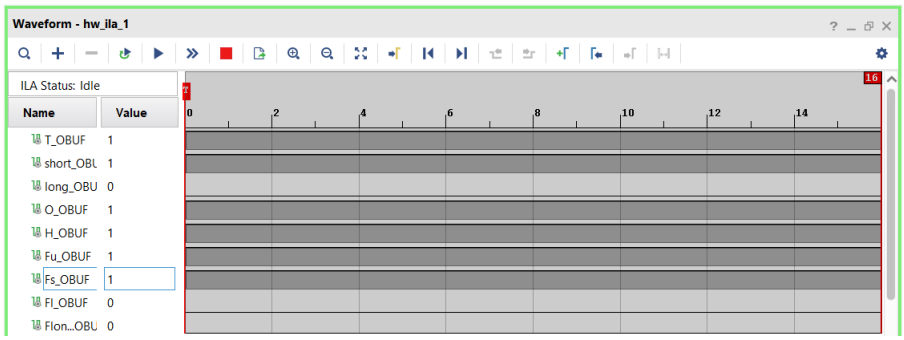
Heat shock response is a eukaryotic cellular stress response replying excessive temperature rise, heavy metal toxicity, associated diseases and related noxious stimulus [206, 207]. Proteins ($prot$) of a cell are misfolded under these stresses and transformed to misfolded proteins (mfp). The primary function of heat shock response of eukaryotes is to produce heat shock proteins (hsp) working as chaperones for mfp to refold them. hsp encoded genes, hse, take in charge of this response process while transcription of these genes are inter-mediated by trimers of heat shock factors (hsf_3). hsf_3 binds to promoter-site of hse, generating $hsf_3 : hse$. Consequently, hsp synthesis is triggered to produce more hsp until a high level to refold $prot$. Then heat shock response ceases [208, 209]. On the other side, hsp binds with hsf to form $hsp : hsf$. When the temperature ascends again, $hsp : hsf$ will be broken to release hsfs which polymerize hsf_3, so the heat shock response restarts [35]. A Petri-net based formalization and simulation of Heat shock responses are proposed in [210].

A heat shock response reaction system $\mathcal{R}_{hsr} = (B_3, A_3)$ is proposed in [35] where B_3 constitutes of entities in Table 4.7 and A_3 consists of reactions in the same table. The interactive processes without temperature stress (at 37°C) and with temperature stress (at 42°C) are listed in Table 4.8 and 4.9 respectively. $\mathcal{R}_{hsr}$ with the two interactive processes are modeled as two distinct RTL models, $\mathcal{R}_{hsr1}$ and $\mathcal{R}_{hsr2}$, for their input/output ports are a little different.

The expected waveform and RTL model behavioral simulation waveform of Table 4.8 and 4.9 are portrayed in Figure 4.11 and A.6. I/O planning of two RTL models of $\mathcal{R}_{hsr1}$ and $\mathcal{R}_{hsr2}$ are shown in Figure A.3. As the clock period is also set as 5 ns

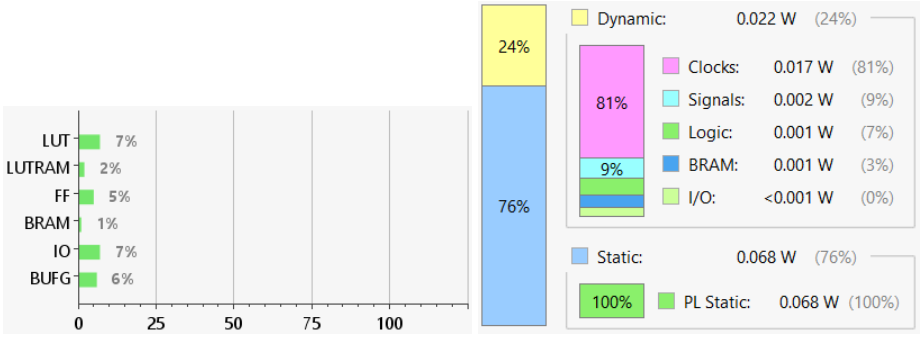


(a) Hardware debug of $\mathcal{R}_{ifl1}$ model.



(b) Hardware debug of $\mathcal{R}_{ifl2}$ model.

FIGURE 4.9: Hardware debug of two intermediate filaments RSs.



(a) Hardware resources dissipation. (b) power consumption.

FIGURE 4.10: Hardware resources dissipation and power consumption of $\mathcal{R}_{ifl1}$.

TABLE 4.7: Reactions of heat shock response RS $\mathcal{R}_{hsr} = (B_3, A_3)$.

No.	RS reactions
(17)	({hsf},{hsp},{hsf_3})
(18)	({hsf,hsp,mfp},{d_i},{hsf_3})
(19)	({hsf_3},{hse,hsp},{hsf})
(20)	({hsf_3,hsp,mfp},{hse},{hsf})
(21)	({hsf_3,hse},{hsp},{hsf_3:hse})
(22)	({hsf_3,hse,hsp,mfp},{d_i},{hsf_3:hse})
(23)	({hse},{hsf_3},{hse})
(24)	({hse,hsf_3,hsp},{mfp},{hse})
(25)	({hsf_3:hse},{hsp},{hsf_3:hse,hsp})
(26)	({hsf_3:hse,hsp,mfp},{d_i},{hsf_3:hse,hsp})
(27)	({hsp,hsf},{mfp},{hsp:hsf})
(28)	({hsp:hsf,stress},{nostress},{hsp,hsf})
(29)	({hsp:hsf,nostress},{stress},{hsp:hsf})
(30)	({hsp,hsf_3},{mfp},{hsp:hsf})
(31)	({hsp,hsf_3:hse},{mfp},{hsp:hsf,hse})
(32)	({prot,stress},{nostress},{prot,mfp})
(33)	({prot,nostress},{stress},{prot})
(34)	({hsp,mfp},{d_i},{hsp:mfp})
(35)	({mfp},{hsp},{mfp})
(36)	({hsp:mfp},{d_i},{hsp,prot})

TABLE 4.8: Interactive process 1 (5 steps) of $\mathcal{R}_{hsr}$ at 37°C.

Step	C_i	D_i	W_i	Applicable reactions
0	{hsf,prot,hse,nostress}	$\emptyset$	{hsf,prot,hse,nostress}	(17),(23),(33)
1	{nostress}	{hsf_3,prot,hse}	{hsf_3,prot,hse,nostress}	(21),(33)
2	{nostress}	{hsf_3:hse,prot}	{hsf_3:hse,prot,nostress}	(25),(33)
3	{nostress}	{hsp,hsf_3:hse,prot}	{hsp,hsf_3:hse,prot,nostress}	(31),(33)
4	{nostress}	{hsp:hsf,hse,prot}	{hsp:hsf,hse,prot,nostress}	(23),(29),(33)
5	{nostress}	{hsp:hsf,hse,prot}	{hsp:hsf,hse,prot,nostress}	(23),(29),(33)

in the timing constraint, clock period of two post implementation timing simulations is 4 ns and 6 ns, shown in Figure A.4, so that the mean time can be used as elapsed time of 5 ns clock implementation. Because the reactions and number of interactive process step of two heat shock response RTL models are the same, only post implementation timing simulation of $\mathcal{R}_{hsr1}$ (Table 4.8) is given in Figure A.4, so as hardware resource dissipation and power consumption (Figure A.5). Hardware debug of two heat shock response RS is given in Figure4.12. The computation speed is also 2×10^8 step per second (200 MHz).

The RS of ErbB receptor signal transduction in human mammary epithelial cells [148] is a large scale RS having 246 entities 6720 reactions. It was implemented with the same way presented here in the same FPGA at the speed of 10^8 steps per second. Comparing with the speed achieved on the GPU-based simulator [145], the speedup is 2.5×10^5. Please refer to [198] for more details.

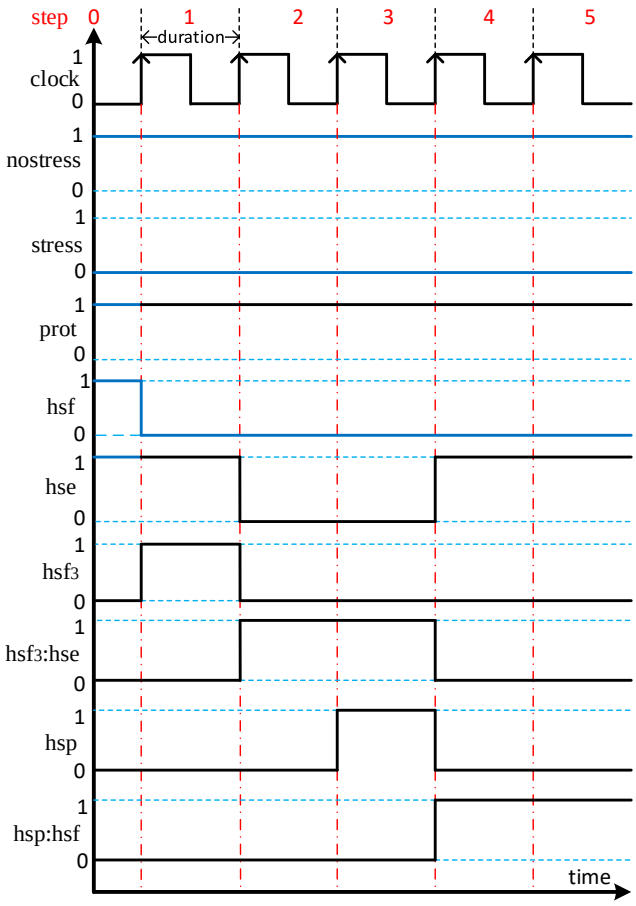
4.4 FPGA implementation of reaction system binary counter

4.4.1 Reaction system binary counter design

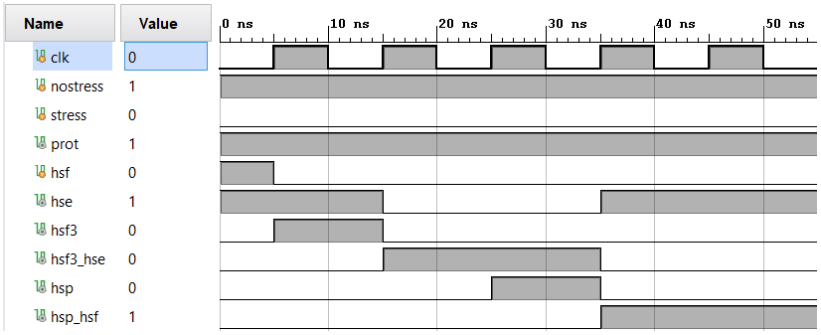
From precedent studies, the "presence 1 absence 0" representation of a entity is a key notion in the design of reaction systems and their FPGA implementations. The

TABLE 4.9: Interactive process 2 (5 steps) of $\mathcal{R}_{hsr}$ at 42°C.

Step	C_i	D_i	W_i	Applicable reactions
0	{hse,prot,$hsp:hsf$,stress}	$\emptyset$	{hse,prot,$hsp:hsf$,stress}	(23),(28),(32)
1	{stress}	{hse,hsp,hsf,prot,mfp}	{hse,hsp,hsf,prot,mfp,stress}	(18),(23),(33),(35)
2	{stress}	{prot,mfp,$hsp:mfp$,hsf_3,hse}	{prot,mfp,$hsp:mfp$,hsf_3,hse,stress}	(21),(32),(35),(36)
3	{stress}	{hsp,prot,mfp,$hsf_3:hse$}	{hsp,prot,mfp,$hsf_3:hse$,stress}	(26),(32),(34)
4	{stress}	{hsp,prot,mfp,$hsf_3:hse$,$hsp:mfp$}	{hsp,prot,mfp,$hsf_3:hse$,$hsp:mfp$,stress}	(26),(32),(34),(36)
5	{stress}	{hsp,prot,mfp,$hsf_3:hse$,$hsp:mfp$}	{hsp,prot,mfp,$hsf_3:hse$,$hsp:mfp$,stress}	(26),(32),(34),(36)

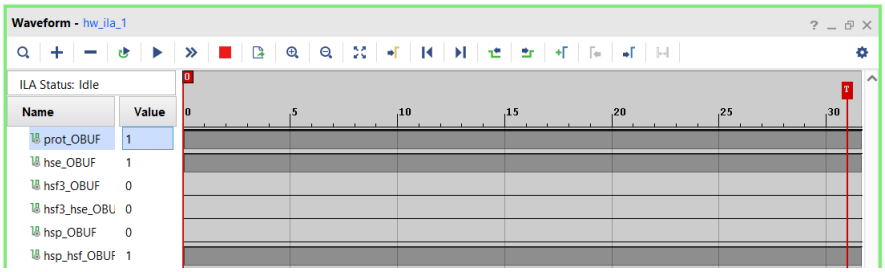


(a) The expected waveform of interactive process listed in Table 4.8. The blue solid lines indicate context entities.

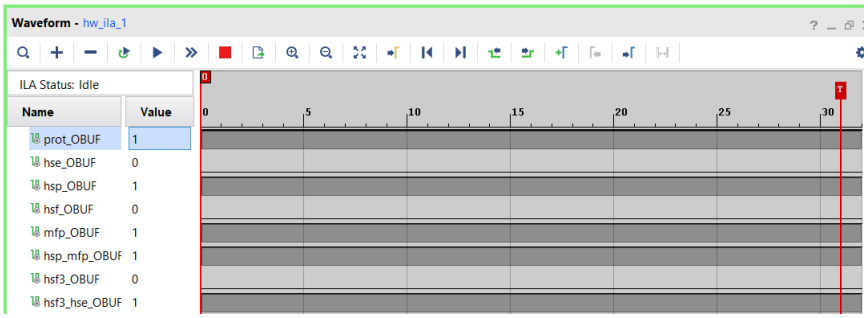


(b) Vivado behavioral simulation waveform of $\mathcal{R}_{hsr1}$ RTL model with interactive process listed in Table 4.8. The clock perid is set as 10 ns.

FIGURE 4.11: Waveform comparison of interactive process 1 of $\mathcal{R}_{hsr} = (B_3, A_3)$.



(a) Hardware debug of $\mathcal{R}_{hsr1}$ model.



(b) Hardware debug of $\mathcal{R}_{hsr2}$ model.

FIGURE 4.12: Hardware debug of two heat shock response RSs.

binary form (1 and 0) of this representation induces the construction of quantitative models, such as a binary counter [25, 30]. Nonetheless, "1 presence 0 absence", not the vice versa, is the underlying idea to contrive a counter. According to this idea, a 4-bit binary number 1010 is denoted from most significant bit (msb) to least significant bit (lsb) by $\{b_4,b_2\}$ for bit four and bit two are 1s so they should present and bit three and bit one are 0s so absent. It is easy to apply this notation to n-bit numbers which $01010\ldots011$ is expressed as $\{b_{n-1}, b_{n-3}, b_2, b_1\}$. b_0 is used as a successor function of adding 1 to the binary number with b_0 attached to, for instance, $\{b_4,b_2,b_0\}$ means adding 1 to 1010.

For a n-bit binary number, assume the i-*th* bit $b_i = 1$ and the j-*th* bit $b_j = 0$, $0 \leq j < i \leq n$, then $b_i = 1$ holds if b_0 is attached. This fact can be reflected by the set of reactions $r_ij = (\{b_i\}, \{b_j\}, \{b_i\})$ where $0 \leq j < i \leq n$. If $b_i = 0$, all the bits lower than i are 1s with b_0 attached, then $b_i = 1$ and all the bits lower than i become 0s after the augment of 1. This situation is represented by reaction set $a_i = (\{b_0, b_1, \ldots, b_{i-1}\}, \{b_i\}, \{b_i\})$ where $1 \leq i \leq n$. With these two reaction sets, a reaction system performing binary counter function can be built as $\mathcal{R}_{bc} = (B_n, A_n)$ where $B_n = \{b_0, b_1, \ldots, b_n\}$, $A_n = \{r_{ij} | 0 \leq j < i \leq n\} \cup \{a_i | 1 \leq i \leq n\}$.

Taking n=4 as an example, the reactions of binary counter is listed in Table 4.10. To implement this counter into FPGA, reactions are transformed to Boolean expressions, as shown in Equation 4.7 which can be simplified to Equation 4.9. $\oplus$ stands for XOR operation. Suppose a n-bit number E composed of $\{b_n,b_{n-1},\ldots,b_1\}$, Algorithm 1 is designed to generalize Formula 4.9.

$$\begin{cases} b_1 = (b_1 \wedge \overline{b_0}) \vee (\overline{b_1} \wedge b_0) \\ b_2 = (b_2 \wedge \overline{b_0}) \vee (b_2 \wedge \overline{b_1}) \vee (\overline{b_2} \wedge b_1 \wedge b_0) \\ b_3 = (b_3 \wedge \overline{b_0}) \vee (b_3 \wedge \overline{b_1}) \vee (b_3 \wedge \overline{b_2}) \vee (\overline{b_3} \wedge b_2 \wedge b_1 \wedge b_0) \\ b_4 = (b_4 \wedge \overline{b_0}) \vee (b_4 \wedge \overline{b_1}) \vee (b_4 \wedge \overline{b_2}) \vee (b_4 \wedge \overline{b_3}) \vee (\overline{b_4} \wedge b_3 \wedge b_2 \wedge b_1 \wedge b_0) \end{cases} \tag{4.7}$$

TABLE 4.10: RS reactions of $\mathcal{R}_{bc}$.

No.	RS reactions
(r_{10})	$(\{b_1\},\{b_0\},\{b_1\})$
(r_{20})	$(\{b_2\},\{b_0\},\{b_2\})$
(r_{30})	$(\{b_3\},\{b_0\},\{b_3\})$
(r_{40})	$(\{b_4\},\{b_0\},\{b_4\})$
(r_{21})	$(\{b_2\},\{b_1\},\{b_2\})$
(r_{31})	$(\{b_3\},\{b_1\},\{b_3\})$
(r_{41})	$(\{b_4\},\{b_1\},\{b_4\})$
(r_{32})	$(\{b_3\},\{b_2\},\{b_3\})$
(r_{42})	$(\{b_4\},\{b_2\},\{b_4\})$
(r_{43})	$(\{b_4\},\{b_3\},\{b_4\})$
(a_1)	$(\{b_0\},\{b_1\},\{b_1\})$
(a_2)	$(\{b_0,b_1\},\{b_2\},\{b_2\})$
(a_3)	$(\{b_0,b_1,b_2\},\{b_3\},\{b_3\})$
(a_4)	$(\{b_0,b_1,b_2,b_3\},\{b_4\},\{b_4\})$

$$\begin{cases} b_1 = (b_1 \wedge \overline{b_0}) \vee (\overline{b_1} \wedge b_0) \\ b_2 = (b_2 \wedge \overline{b_1 \wedge b_0}) \vee (\overline{b_2} \wedge b_1 \wedge b_0) \\ b_3 = (b_3 \wedge \overline{b_2 \wedge b_1 \wedge b_0}) \vee (\overline{b_3} \wedge b_2 \wedge b_1 \wedge b_0) \\ b_4 = (b_4 \wedge \overline{b_3 \wedge b_2 \wedge b_1 \wedge b_0}) \vee (\overline{b_4} \wedge b_3 \wedge b_2 \wedge b_1 \wedge b_0) \end{cases} \tag{4.8}$$

$$\begin{cases} b_1 = b_1 \oplus b_0 \\ b_2 = b_2 \oplus (b_1 \wedge b_0) \\ b_3 = b_3 \oplus (b_2 \wedge b_1 \wedge b_0) \\ b_4 = b_4 \oplus (b_3 \wedge b_2 \wedge b_1 \wedge b_0) \end{cases} \tag{4.9}$$

Algorithm 1: reaction system binary counter.

Input: 1-bit variable b_0
Output: the times that $b_0 = 1$

```
for i = 0; i < n; i++ do
    t = e_0; for j = 0; j < i; j++ do
        t = E[j] ∧ t;
    end
    E[i] = E[i] ⊕ t;
end
```

4.4.2 UART-RS counter design and implementation

FPGA implementation of RS binary counter is conducted based on Algorithm 1 for its parametrization of bit number. To validate the correctness of RS counter, and to push forward one step closer to potential applications, universal asynchronous receiver/transmitter (UART) module is devised and added to RS counter module so that b_0 signal sending from host computer can be received and counted by the RS counter hardened in FPGA. The counting result can be transmitted back to host computer by UART module hardened in FPGA to display directly, making hardware debug procedures unnecessary.

UART is a serial communication protocol without common clock between transmitter and receiver that are wired together. Their clocks are independent but must have the same frequency so that what is sending can be received properly. As a widely used serial ports, UART is a default device of computers. So only FPGA

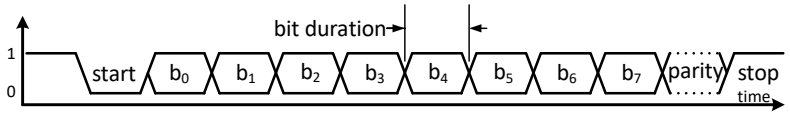


FIGURE 4.13: UART data frame with 8 bits data, 1 bit EOF. Parity bit is optional.

UART should be designed and implemented to communicate with host computer. As a serial protocol, what transmitted is a train of single bit data. These binary bits are packaged into a frame with one bit of 0 as the start of frame (SOF) and one or two bits of 1 as the end of frame (EOF). The number of bit ranges from 5 to 8, followed by one optional parity bit [211]. The UART data frame is captured in Figure 4.13. In this research, the 10-bit data frame consisting of 1 bit SOF, 8 bits data, 1 bit EOF and no parity bit, is used. $b_0 = 1$ denotes the successor function of adding 1 is performed so the value in the counter will augment by 1. $b_0 = 0$ means that the successor function will not happen.

The RTL model of RS counter has three 1-bit input ports: *clk* which is the clock port, ib_0 and *state* which controls the shifting of work mode between idle and busy. The output port is a 8-bit variable E whose initial value is 0. E[7]$\sim$E[0] correspond to $b_8 \sim b_1$ so at the beginning E[7]=E[6]=...=E[0]=0. The experiment is arranged as follows: 32 frames of 8-bit data in which one bit is 1 and the others are 0s are sent from host computer UART transmitter to RS counter. There are thirty two 1s in total passing through ib_0 port so the successor function of adding 1 should be executed 32 times, resulting in that the value of E is 32 which is transmitted from FPGA to host computer to display. The transmitter sends data bits sequentially so it seems that these data frame can be input to RS counter directly to count. Unfortunately it is wrong to do so. Bear in mind that the EOF of each data frame is also 1 and RS counter has no mechanism to distinguish the EOF 1 from data bit 1. The result will be 64 instead of 32 for all the EOF 1s are counted as well.

To avoid this mistake, data frames are received by the UART receiver R_1 and stored in 32 8-bit registers successively. A special transmitter T_1 whose EOF is also 0 sends these data serially to RS counter to count the data bit 1 which denotes the attachment of successor function trigger b_0. After counting completed, result is sent back to host computer by normal transmitter T_2 at the same baud rate of reception from host computer. Transmission/reception rate of host computer is set as 9600 baud (this is done by setting properties of COM port). To transmit/receive data at this speed, the first thing to do is synthesizing a clock whose frequency is 9600 Hz from master clock whose frequency is set to 100 MHz. Direct digital synthesis (DDS) [212, 213] is a frequency synthesis technique playing a important role in the digital systems [214]. DDS tuning equation is given in Formula 4.10 [215], where f_o is the output synthesized frequency, f_s is the master clock frequency, K is the binary tuning word, N is the bit length of phase accumulator. K is a integer with the range of $[2^0, 2^{N-1}]$.

$$f_o = \frac{f_s \times K}{2^N} \tag{4.10}$$

If a specific frequency is desired, its corresponding tuning word can be computed by Formula 4.11. K is accumulated in the phase accumulator register PA. The median value of $2^N - 1$ signified by M is used as threshold so that if PA<M then assign 0 otherwise assign 1 to the synthesized clock to output. However, synthesized clocks are not allowed to drive circuits since it is difficult to synthesize these circuits

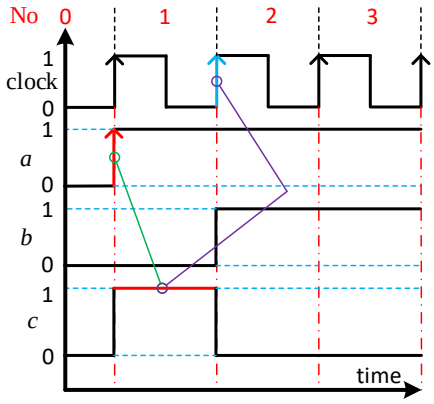


FIGURE 4.14: Rising edge detection. The edge detection method can generate a high level signal lasting for 1 clock cycle after the red rising edge. It is used in combination with the clock rising edge highlighted in blue as the trigger signal for other operations.

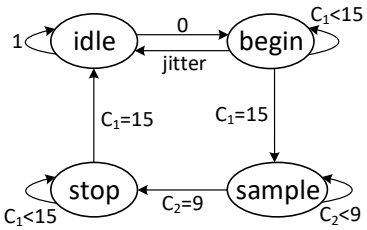


FIGURE 4.15: State transition diagram of UART receiver R_1.

and their stability cannot be guaranteed [216]. To handle this adversity, the trick of edge detection is utilized to generate a high level which has the same frequency of the synthesized clock, as shown in Figure 4.14 where a high level lasting one clock period is produced after the rising edge highlighted in red color. This high level together with the clock rising edge in blue can drive circuits at the synthesized clock frequency. In shortly, transforming edge trigger of synthesized clock to level trigger is a secure solution for driving circuits by synthesized clocks.

$$K = \frac{f_o \times 2^N}{f_s} \tag{4.11}$$

To sample data at the most stable phase of the data duration, the frequency of UART receiver R_1 is expedited to 9600×16. So in a period of host computer transmitter clock, there are 16 periods of R_1 clock. Perform positive edge detecting of R_1 clock to get 16 high level lasting one period of master clock which is 10^8 Hz. A counter C_1 counts the number of high levels from 0 to 15. When the value of C_1 is 7, the time moment is the middle of the data duration. Sampling at this moment can get the most stable value for it is far away from two endpoints where data hopping happens. When $C_1 = 15$, a bit counter C_2 with 0 as initial value adds 1 to itself. During $C_2 = 0$ and $C_2 = 9$, no samplings are executed to skip SOF and EOF of each data frame. R_1 is implemented as a state machine with 4 states as illustrated in Figure 4.15.

Data frames received by R_1 are stored in thirty two 8-bit registers from a_0 to a_{31} serially. Once a register has fully loaded, a high level signal $t1_d$ generated by edge detection arises and a counter C_3 counts the number of this signal from 0 to 31. The high level signal drives T_1 to begin transmitting data to RS counter. T_1 sends 10 bits

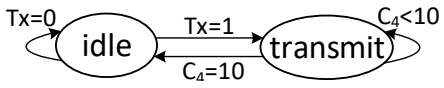


FIGURE 4.16: State transition diagram of UART transmitter T_1.

during its duty cycle for a data frame comprises 10 bits of binary data, with 0 as the EOF rather than 1 for the reason stated above. Counter C_4 counts the number of bits sent by T_1 which is implemented as a 2-state machine depicted in Figure 4.16 where `Tx` indicates transmitting signal. Because host computer sends data at 9600 baud, RS counter should compute at 960 baud for the 10-bit frame (including 1 bit of SOF, 1 bit of EOF and 8 bits data) is sent back at 9600 baud too. As a consequence, T_1 works at 960 baud (this means the data duration of a bit is the period of 960 baud) to feed RS counter, i.e., once one bit of data comes in through ib_0 port, perform reactions involved. The drive signal of T_1, $t1_d$, has a frequency of 96 baud since there are 10 bits between two high levels of $t1_d$. The result is transmitted back to host computer by a normal UART transmitter T_2 with 1 as EOF, at the rate of 9600 baud.

It is emphasized that the metric baud can be substituted by Hz. This assertion is verified by experiments conducted. Baud rate can be improved easily for clock synthesis is based on parametric design method. Note that the precision of UART communication is not very high without parity bit so if the baud rate is too high, the error would be large. RTL block diagram of UART-RS counter is presented in Figure 4.17. Behavioral simulation of RTL model is shown in Figure 4.18 in which the result stored in E is 32. This value is exactly correct for there are thirty two 1s in all the data frames. Communication experiment result between host computer and FPGA is shown in Figure 4.19 where numbers are represented in hexadecimal so that 20 denotes decimal 32. This result validates the correctness of RS counter and its UART communication component.

4.5 Wrap-up

The relationship between reaction systems and synchronous digital circuits is investigated to lay the foundation of FPGA implementation of reaction systems. Reactions are represented as Boolean expressions for the underlying qualitative property of threshold supply assumption. Duration of entities is interpreted as clock period to present no permanency assumption. Two small scale biological models modeled by RS, i.e., intermediate filaments self-assembly and heat shock response RSs are implemented in FPGA, achieving a computing speed of 2×10^8 step per second for the high efficiency of executing logic formulas. The binary counter which is a quantitative model constructed by a reaction system is implemented in FPGA as well. To verify the correctness of RS counter, the UART communication component which coordinates with the computing speed of RS counter is devised. These two parts are connected as a entirety and implemented in FPGA. The communication experiment shows that UART-RS counter works as expected.

Essentially, a reaction system is a set of reactions. With the input of context set C_i, some reactions are facilitated so applicable and some are inhibited. All the applicable reactions are executed concurrently, without conflict for competing common entities. The absence of a entity is represented as logic 0 and presence logic 1 to conform with the representation of reactions. These logic expressions are put in the *always* block of Verilog HDL and non-blocking assignment operator $Leftarrow$ is used to perform parallel assignment in the sequential construct. The duration of a interactive process

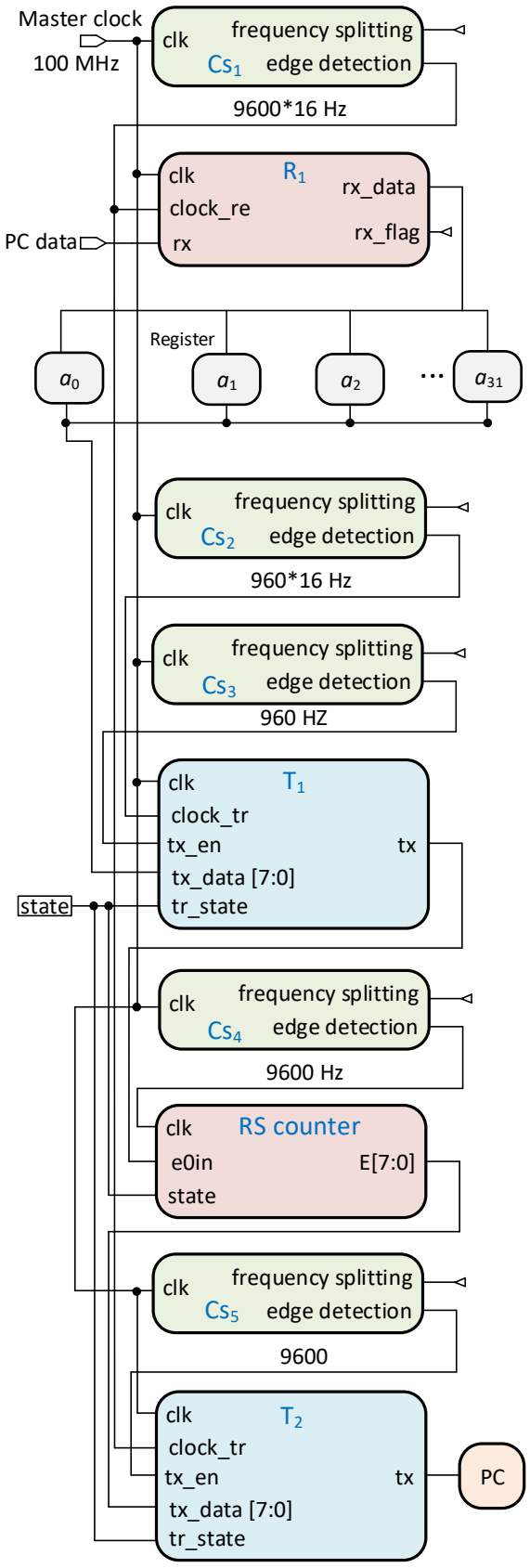


FIGURE 4.17: RTL model block diagram of UART-RS counter. Cs_1 to Cs_4 are clock synthesis module producing desired clock frequencies.

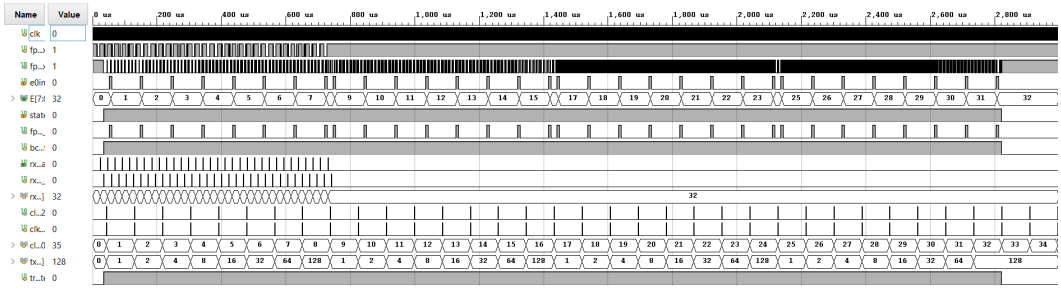


FIGURE 4.18: Behavioral simulation of UART-RS counter RTL model.

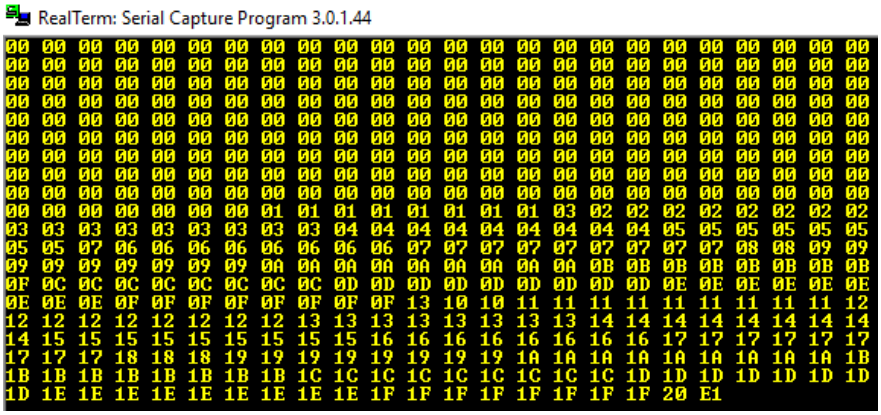


FIGURE 4.19: Communication experiment between host computer and FPGA hardened UART-RS counter. The last E1 is an end mark signifying the completeness of counting.

step is indicated by the clock period of FPGA. Form researches in this chapter, it is shown that quantitative computing models can also implemented on FPGA with considerable speed.

Quantitativeness and qualitativeness are different facets which are related tightly in digital circuits. A computer can output quantitative outcomes while digital circuits inside CPU perform Boolean algebra which is ”qualitative“. Arithmetic logic units compute results based on qualitative other than quantitative laws. For instance, the function of digital adder is realized by XOR gates. There is no quantitative computing at all. In the later chapters, fixed point and floating point arithmetic operations are widely used in NPS. These quantitative operations are qualitative in essentially. This fact tells us that qualitative models have special potentialities which should be investigated deeply.

Chapter 5

Theoretical investigations of numerical P systems

Numerical P systems introduced in [217] is a variant of P systems very different from the standard model. Instead of objects and rewriting rules it features real-valued variables which are updated in discrete time steps using a set of equations. This particularity is very interesting for applications in the area of control theory, as usually a control described using a set of differential equations, which in many cases can be translated to the NPS equation set. In order to be applicable to practical case studies, NPS model was extended to Enzymatic Numerical P systems (ENPS) [218] that allowed much more complex behaviors in the corresponding equations.

There are many theoretical researches about (E)NPS. The computational power and special attributes of NPS and ENPS are investigated before their applications. NPSs with migrating variables were studied in [219] while NPSs with production thresholds were analyzed in [220]. Universal ENPSs with small number of enzymatic variables are discussed in [221]. String languages generated by sequential NPSs were deliberated in [222]. In [223], NPSs with production thresholds was considered. In [224], four recent research topics on numerical P systems were summarized. In [225], universality of ENPSs was examined. Enzymatic numerical P systems using elementary arithmetic operations were investigated in [226]. The computational power of ENPSs working in the sequential mode was studied in [227]. A parallel bio-inspired framework for numerical calculations using ENPS with an enzymatic environment was constructed in [228]. ENPSs for basic operations and sorting were designed in [229]. The ways of how to improving the universality results of ENPS were researched in [230]. The pole balancing problem with ENPS was discussed in [231].

In this chapter, the (E)NPS is extended to a new model called generalized numerical P systems (GNPS). The main idea behind this extension is to provide a theoretical background allowing us to build custom parallel hardware architectures using Field-Programmable Gate Array (FPGA). It is studied what operations can be efficiently performed in hardware and restricted the GNPS architecture to be a series of rules implying that the dynamics of the system is described using equations written in Presburger arithmetic. This allows in turn a very efficient translation to Verilog HDL used for FPGA circuits design. To assist this translation we developed a compiler that translates GNPS to Verilog HDL. This allows to simplify the design process and to rapidly develop real hardware prototypes. Moreover, it turns out that there is a tight link between GNPS, sequential circuits [232], Mealy/Moore automata [195, 200] and synchronous programming languages like Esterel [233, 234, 235] and Lustre [236, 237].

5.1 Numerical P system and enzymatic numerical P system

The formal definition of a numerical P system (NPS) was presented in 2006 [217] and paraphrased here for a better understanding. While the motivation of initiating NPS is to deal with potential applications in economic domain in which involves large scale real number variables, NPS was put to use in robot motion control at first [238].

Definition 5.1. *A numerical P system is the construct*

$$\Pi = (m, H, \mu, (Var_1, Pr_1, Var_1(0)), \ldots, (Var_m, Pr_m, Var_m(0)))$$

where

1. $m > 0$ *is the number of membranes (the system degree);*
2. H *is the membrane label set storing the labels of each membrane;*
3. μ *is the membrane structure;*
4. Var_i, Pr_i, $Var_i(0)$ *are variables, programs and initial values of variables in membrane* i, $1 \leq i \leq m$.

Comparing to multiset rewriting rules in classical symbol object P systems, the rules in a NPS named as *programs* are quite unusual with production function in the left hand side and repartition protocol in the right hand side, taking on the form

$$P_{li} : F_{li}(x_{1i}, \ldots, x_{k_i i}) \rightarrow c_{l1}|v_1 + \cdots + c_{ln_i}|v_{n_i}$$

where $x_{1i}, \ldots, x_{k_i i}$ are variables in membrane i, F_{li} is the production function, $1 \leq l \leq n$ is the number of programs in membrane i. $v_1, \ldots, v_{n_i}$ are variables in membrane i or in the upper immediate or lower immediate membrane of membrane i. $c_{l1}, \ldots, c_{ln_i}$ are the distribution coefficients. The result of production function is distributed as Formula 5.1.

$$\begin{cases} v_1 = \frac{c_{l1} \times F_{li}(x_{1i}, \ldots, x_{k_i i})}{\sum_{t=1}^{n_i} c_{lt}} \\ \vdots \\ v_{n_i} = \frac{c_{ln_i} \times F_{li}(x_{1i}, \ldots, x_{k_i i})}{\sum_{t=1}^{n_i} c_{lt}} \end{cases} \tag{5.1}$$

If a variable only appears in a production function, after the execution of current computation step, the former value is consumed so its new value is 0. For variables arising in both sides of the same program, their former values are overwritten by newly distributed values according to repartition protocols. If a variable appears merely in several repartition protocols of different programs, its new value is the sum of all distributed values plus its former value. On condition that variables arise in both sides of several programs, their new value after the computation step is the sum of all distributed values that overwrite their former values.

Since every program is applicable and executes in parallel, in case of multiple programs populating in one membrane, one program is selected randomly to execute. This derivation mode is called min_1 [91]. The random selecting of programs imparts the non-determinism to NPS, bringing on a negative impact on robot motion control for the control law is deterministic. Hence the results of each computation step of NPS should also be definite. For the sake of avoiding this non-determinism

of NPS, only one program is assigned to every membrane to eliminate the random selection process, obtaining a deterministic NPS as a result.

Although the determinism is guaranteed by the one to one correspondence of membranes and programs, the membrane structure tends to be complex and its efficiency declines. This drawback becomes more obvious for modeling sophisticated algorithms which need plenty of membranes to distribute operations. Enzymatic numerical P systems (ENPS) introduced in 2010 [218] can contain multiple programs in one membrane, adopting enzyme-like variables to decide the usability of programs. The application of a program is conditioned to the verification of the minimality condition between the values of *enzyme* (or *enzymatic variable*) and its reactants. For example, the program $2x + y - 1(e \rightarrow)z$ of a ENPS is applicable only if $e > min(x, y)$. Unfortunately, there is no unique definition for this condition – several papers use different ones. Here is the list of most used conditions:

$$e > \min(c(x_{1i}), \ldots, c(x_{k_i i})), \text{ in [218]}$$
$$e > \min(x_{1i}, \ldots, x_{k_i i}), \text{ in [218]}$$
$$e > \min(|x_{1i}|, \ldots, |x_{k_i i}|), \text{ in [239].}$$

The biological base for this criterion is that the concentration of a enzyme should large than that of a reactant. Different with the catalyst used in classical symbol object P systems, enzymes can be consumed or produced, and their amounts matter. For programs without enzymes, these programs are applicable automatically. The definition of ENPS is quoted bellow.

Definition 5.2. *An enzymatic numerical P system is the construct*

$$\Pi = (m, H, \mu, (Var_1, E_1, Pr_1, Var_1(0)), \ldots, (Var_m, E_m, Pr_m, Var_m(0)))$$

where

1. $m > 0$ *is the number of membranes (the system degree);*
2. H *is the membrane label set storing the labels of each membrane;*
3. μ *is the membrane structure;*
4. Var_i, Pr_i, $Var_i(0)$ *are variables, programs and initial values of variables in membrane* i, $1 \leq i \leq m$.
5. E_i, $1 \leq i \leq m$, *are the enzyme variables in membrane* i;

The form of a program with enzyme in a ENPS is little different from NPS programs as shown bellow,

$$P_{li} : F_{li}(x_{1i}, \ldots, x_{k_i i})(e_{j,i} \rightarrow)c_{l1}|v_1 + \cdots + c_{ln_i}|v_{n_i}$$

where $e_{j,i} \in E_i$ (j is the mark number), is the enzyme catalyzing this program. The computation procedures of ENPS is identical to that of the NPS. The *all-parallel* mode [240] by which a set of applicable programs determined by enzymatic variables, instead of a single program (min_1 mode), are executed simultaneously, is the considered mode on ENPS. The function of enzymes greatly simplifies the membrane structures, making ENPS more practicable to model engineering applications.

Comparing to the symbol object P system in *max* mode and NPS in min_1 mode, the ENPS in *all-parallel* mode is a mixed model with quantitative and qualitative

properties: if multiple programs involving a common variable are applicable determined by enzymes, the value of the common variable is copied and assigned to every one in different programs. There is no limits on the copied value of common variable such that different programs having the same variable can use the copied value without conflict. This qualitative attribute of ENPS is turned out to be quite efficient to arrange complicate algorithms of real life applications.

5.2 The relationship between (E)NPS and system of difference equations

5.2.1 From (E)NPS to system of difference equations

Consider the following NPS, also depicted in Figure 5.1, with two membranes nested as follows: $[_1[_2]_2]_1$. Let $Var_1 = \{a, b, f\}$, $Var_2 = \{x, y\}$, $Var_1(0) = (0, 1, 3)$, $Var_2(0) = (0, 1)$. The rules of the system are defined as follows:

$Pr_{1-1} : 4(a + b) \to 1|a + 1|f + 2|x.$
$Pr_{1-2} : 3(x + y) \to 1|b + 1|x + 1|y.$

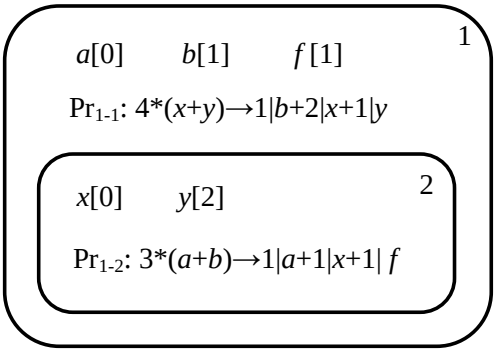


FIGURE 5.1: Target numerical P system. The nested (membrane) structure is represented by a Venn diagram; the variables and the rules are placed in corresponding locations; the initial value of variables follow them in square brackets.

It is not difficult to observe that the corresponding system can be rewritten as following difference equations with initial conditions $a(0) = 0$, $b(0) = 1$, $f(0) = 3$, $x(0) = 0$, $y(0) = 1$.

$$\begin{cases} a(t+1) = a(t) + b(t) \\ b(t+1) = x(t) + y(t) \\ f(t+1) = f(t) + a(t) + b(t) \\ x(t+1) = x(t) + y(t) + 2\,(a(t) + b(t)) \\ y(t+1) = x(t) + y(t) \end{cases} \tag{5.2}$$

This can be also written in a matrix form as follows

$$\begin{bmatrix} a(t+1) \\ b(t+1) \\ f(t+1) \\ x(t+1) \\ y(t+1) \end{bmatrix} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 2 & 2 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} \begin{bmatrix} a \\ b \\ f \\ x \\ y \end{bmatrix} \tag{5.3}$$

In many cases difference equations can be solved analytically using standard methods like matrix diagonalization or generating function. The analytical solution for system defined by Equation (5.2) is given below:

$$a(t) = 2 \times 3^{t-2}, \qquad b(t) = 4 \times 3^{t-2}, \qquad f(t) = 3^{t-1} + 3,$$
$$x(t) = 8 \times 3^{t-2}, \qquad y(t) = 2 \times 3^{t-2} + 1, \qquad t > 1$$

Now let us switch to the ENPS depicted in Figure 5.2, with three membranes nested as follows: $[_1[_2]_2[_3]_3]_1$. Let $Var_1 = \{a, b\}$, $Var_2 = \{E\}$, $Var_3 = \{c\}$, $Var_1(0) = (0, 3)$, $Var_2(0) = (2n)$, $n > 0$, $Var_3(0) = 0$. The set of rules is defined as follows:

$Pr_1 : 2b + 1(E \rightarrow)1|a,$
$Pr_2 : E - 1 \rightarrow 1|E,$
$Pr_3 : 2(c + 1)(E \rightarrow)1|c + 1|b.$

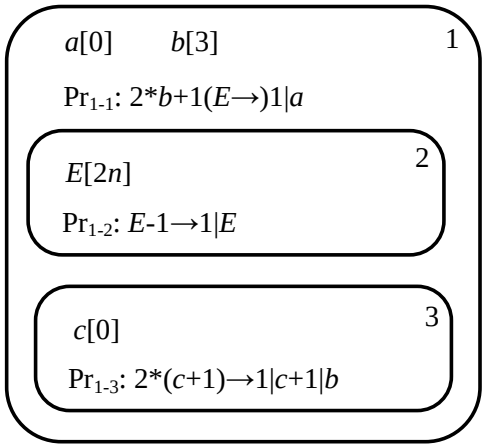


FIGURE 5.2: Target enzymatic numerical P system.

It is not difficult to observe that the corresponding system can be rewritten as following difference equations.

$$\begin{aligned} a(t+1) &= \text{if } E(t) > b(t) \text{ then } 2b(t) + 1 \text{ else } a(t) \\ b(t+1) &= c(t) + 1 \\ c(t+1) &= c(t) + 1 \\ E(t+1) &= E(t) - 1 \\ a(0) = 0, \quad b(0) &= 0, \quad c(0) = 0, \quad E(0) = 2n \end{aligned}$$

This difference equations are equivalent to the equation: $a(t) = \sum_{i=0}^{t} 2i + 1 = t^2$ (for $t \leq n$).

5.2.2 From system of difference equations to (E)NPS

The system of difference equations producing Fibonacci sequence is represented by Formula 5.4 where $f(0) = 0$, $f(1) = 1$. Equation 5.4 can be transformed to a system of difference equations given in Formula 5.5 which contains only (n-1). The NPS carrying out Equation 5.5 is portrayed in Figure 5.3. It has two nested membranes encompassing two equations from Formula 5.5.

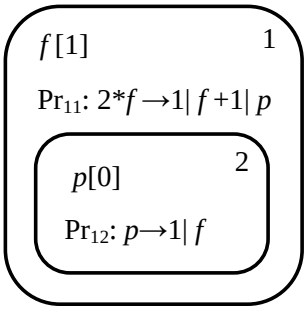


FIGURE 5.3: The numerical P system generating Fibonacci sequence.

$$f(n) = f(n-1) + f(n-2) \tag{5.4}$$

$$\begin{cases} f(n) = f(n-1) + p(n-1) \\ p(n) = f(n-1) \end{cases} \tag{5.5}$$

Consider the projectile motion depicted in Figure 5.4 as another example. This is a continuous model which should be transformed to its difference counterpart at first, as given in Formula 5.6. The corresponding NPS is illustrated in Figure 5.5. Note that S_y is a non-linear continuous function which is approximated by a piecewise-linear function. The benefit of such approximation is order reduction, which can improve computing speed in software simulation and hardware implementation, at the price of a lower precision of results.

$$\begin{cases} v_x(t+1) = v_0 \\ S_x(t+1) = S_x(t) + v_0 \\ v_y(t+1) = v_y(t) + g \\ S_y(t+1) = S_y(t) + v_y(t) + \frac{g}{2} \end{cases} \tag{5.6}$$

As can be seen, there is a correspondence from a (E)NPS to a discrete time series then to a difference system. So a NPS is another form of its corresponding difference system. Difference and differential equations are tightly related. There are standard methods transforming one to another [241, 242]. As a consequence, (E)NPSs can be used to model difference systems originated from differential systems. This is a pretty useful conclusion for practical applications of NPS since most engineering models are characterized by differential systems. With this conclusion, difference system described models can be transformed to (E)NPSs whose advantage are showing quantitative correlations more clear with chemical reaction like rules/programs. On the other side, it shows that NPSs have good adaptability to model many practical situations which may benefit from the inherent parallelism of (E)NPS.

5.3 Binary and unary normal form of (E)NPS

Form the standard form of a (E)NPS program it is indirect to see the distributed portion of a variable in the repartition protocol for there are multiple variables generally. The distributed portion can be shown straightforwardly by introducing *binary* form of programs. Please refer to Chapter 2 for definitions of NPS and ENPS if necessary.

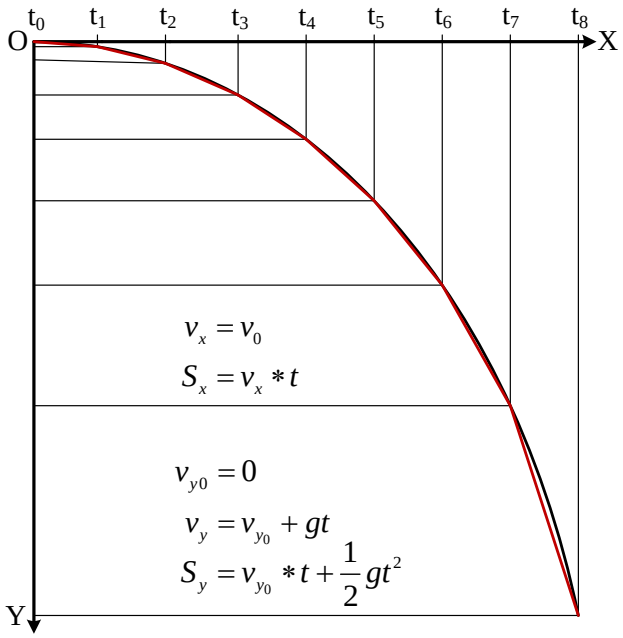


FIGURE 5.4: The projectile motion. The red solid line is the piecewise-linear approximation of the non-linear trajectory.

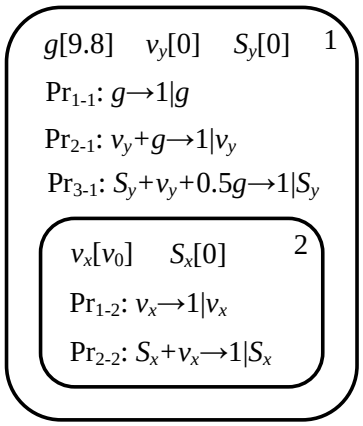


FIGURE 5.5: The NPS corresponding to the system of difference equations given in Equation 5.6.

Definition 5.3. *A (E)NPS is said to be in the binary normal form if all programs are of form*

$$P_{li} : F_{li}(x_{1i}, \ldots, x_{k_i i}) \rightarrow c_s|v_s + (K - c_s)|\lambda, \text{ or}$$
$$P_{li} : F_{li}(x_{1i}, \ldots, x_{k_i i})(e_{li} \rightarrow)c_s|v_s + (K - c_s)|\lambda,$$

for some $i > 0$, $k_i > 1$, $c_s \in (c_{l1}, c_{l2}, \ldots, c_{ln_i})$, $n_i > 1$, $K = \sum_{p=1}^{l} c_p$, $L = K - c_s$ is balancing coefficient, λ is a special dummy variable.

It is obvious that $v_s = \frac{c_s}{K} F_{li}(x_{1i}, \ldots, x_{k_i i})$ is the allocated value form this program. Term *binary* stems from the 2 tuple (c_s, K). A special case is that there is only one variable in repartition protocol, i.e., $l = 1$, resulting in $c_s = K$. Then $L = K - c_s = 0$, so there is no dummy variable λ introduced in the program. This special normal form is called *unary normal* form.

Definition 5.4. *A (E)NPS is said to be in unary normal form if all programs are of form*

$$P_{li} : F_{li}(x_{1i}, \ldots, x_{k_i i}) \to c_s|v_s, \text{ or}$$
$$P_{li} : F_{li}(x_{1i}, \ldots, x_{k_i i})(e_{li} \to)c_s|v_s,$$

where $i > 0$, $k_i > 1$.

If we relax the condition that variables of the production function should be from the same membrane, then it is possible to obtain a stronger result. This allows to combine all rules related to a single variable into one rule by choosing appropriate coefficients. In this case several programs of type

$$F_n(x_{1j}, \ldots, x_{kj}) \to c_n|x + z_n|\lambda$$

can be combined as

$$\sum_{p=1}^{n} \frac{c_p}{c_p + z_p} F_p(x_{1j}, \ldots, x_{kj}) \to 1|x.$$

Comparing to binary normal form, unary normal form has a term $\frac{c_s}{K}$ off and all the value of production function is assigned to one variable. Unary normal form is quite suitable for CPU simulation or FPGA implementation because divider is a resource-power hungry arithmetic unit with long latency. This form makes division unnecessary to improve performance and diminish resource-power consumption. As a consequence, programs are designed to unary normal form as many as possible in applications.

5.4 The relations between symbol-object P system and NPS

Despite the discrepancies in variable type, rule format, and work pattern between symbol object P systems and (E)NPSs, they can be viewed from the quantity prospective to reveal their intrinsic relations to present a in-depth comprehending of both models. A rewriting rule $r_1 : a^3b^2c \to d^2e$ can be decomposed to a set programs given in Formula 5.7, where k_1 is a non-negative integer bounded by derivation mode, denoting the instance (number of used times) of r_1. Once r_1 applied, all the programs comprising it are put to use concurrently, although there are no enzymatic variables to catalyze these programs like that in ENPS. This parallel utilizing also violates program usage mode in NPS where only one program is selected randomly to use. Therefore, it is supposed that rewriting rules of a P system are applied in the light of a special mode in which programs are utilized in parallel, without any extrinsic conditions or constraints.

$$r_1 = \begin{cases} Pr_{11} : a - k_1 \times 3 \to 1 \mid a \\ Pr_{21} : b - k_1 \times 2 \to 1 \mid b \\ Pr_{31} : c - k_1 \times 1 \to 1 \mid c \\ Pr_{41} : d + k_1 \times 2 \to 1 \mid d \\ Pr_{51} : e + k_1 \times 1 \to 1 \mid e \end{cases} \tag{5.7}$$

Along this line, the P system portrayed in Figure 2.2 is adapted as shown in Figure 5.6. The consumption and production of the same object in the program set of a rewriting rule is not offset because the constant coefficients in programs are used to compute the maximal instance of a rule. They can be countervailed

during software simulation or hardware implementation to simplify computations. Procedures calculating maximal instance of a rule are summarized below.

1. perform a serial of divisions $(\frac{N^{obi}}{N^{obi}_{obml}})_t$, $t = 1, 2, \ldots$, where N^{obi} is the number of object i in the object multiset of a membrane, N^{obi}_{obml} is the number of object i in the obm_l of a concerned rule, t is number of object types in obm_l;

2. compute $min((\frac{N^{obi}}{N^{obi}_{obml}})_t)$, where function $min()$ give the minimum number which is the maximal instance the rule can be applied.

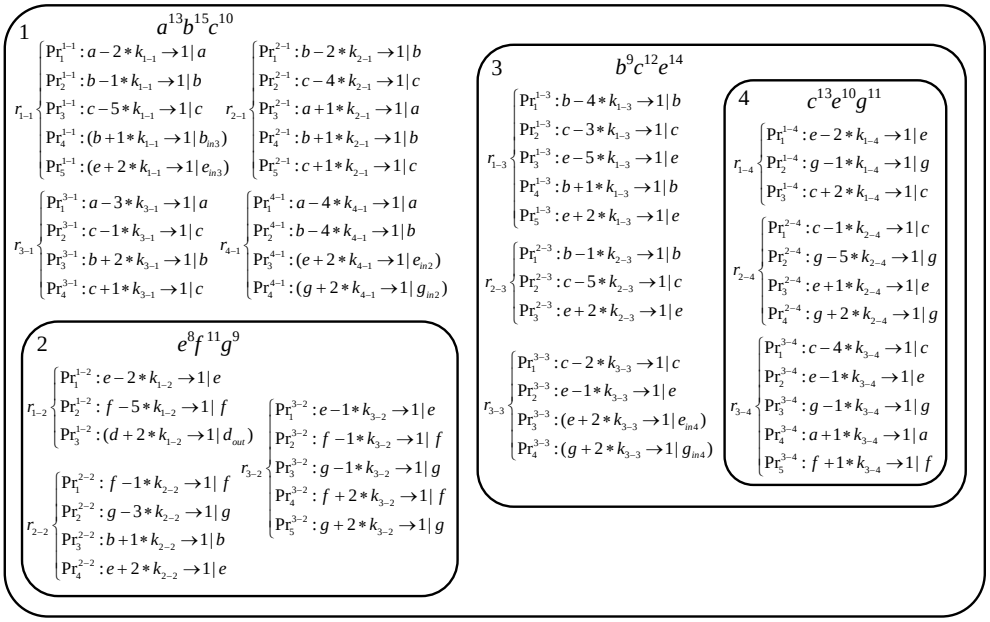


FIGURE 5.6: The symbol object P system described in NPS form. A rewriting rule is translated in a set of programs in which programs involving subtraction denote obm_l and programs involving addition signify obm_r. Programs in parentheses are executed in membranes indicated by their subscripts, not in membranes containing them.

Programs of a object rewriting rule involve merely one type of object while their counterparts in a NPS embody several types of objects. The entire value of a variable is allocated to one program in a NPS, while multiplicities of objects can be shared by different rules in a symbol object P system. This fact means that object multisets are detachable while variables are not. The separability of object multisets gives rise to the multiple feasible solutions of *max* mode, resulting in the difficulty to get all the solutions. The indivisibility of variables refrains NPS from the adversity to compute such feasible solutions. Instead, just select a program randomly, assigning all the values of involved variables to this program to evolve configurations. What is more significant is that these relations are the foundation of software simulation and hardware implementation of symbol object P systems for the execution of rules is realized by performing corresponding programs as shown in Figure 5.7. As can be seen, rule instances are calculated by virtue of the form of rules rather than program form, for the latter one lost some information about usability.

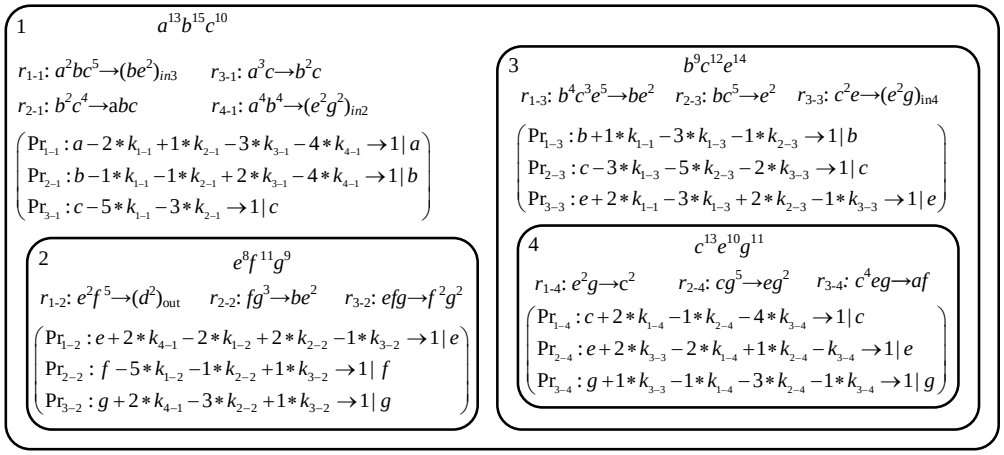


FIGURE 5.7: The effect of rules is equivalent to the set of programs in each membrane, where k_{i-j} denotes instance of rule r_{i-j}. Once k_{i-j} have computed, these programs are used to update multiplicity of objects.

5.5 Generalized Numerical P Systems

In this section, the generalization of (E)NPS called Generalized Numerical P System (GNPS), which has some interesting properties helpful for the hardware implementation of the model, is introduced. As a starting point, the notion of the membrane controller [238] is recalled. Hence from the beginning, the dynamics of the model other than the final result of the computation is paid more attention to. This naturally leads to the inclusion of the concept of dedicated input and output variables. The functioning of the system supposes that input variables are read-only and can be updated by an external entity at each step. The output variables are write-only and an external entity may use their values at each step. Such a definition allows to efficiently build controllers based on GNPS, without using any additional tools or mechanisms to pass the values and start/stop the computation.

From the structural point of view, a structural abstraction intermediate between a tree-based structure and a flattened system (more precisely a hypergraph structure) is employed, being the equivalent of the network of cells [91] in NPS. This allows to have the notion of the locality (useful for hardware implementation as it can trigger the use of neighbor cells), but does not impose the strong restriction of a tree structure — some examples of NPS based membrane controllers spend an enormous amount of time for the data propagation because of the imposed tree structure. Concretely, this allows production functions to contain variables defined in a different membrane and also the repartition protocol may involve variables from any combination of membranes.

The main difference of GNPS with respect to previous models is a new type of rules that generalize all previous ones. This comes from the observation that rules of ENPS are rather limited (and also have a poorly defined semantics). The ENPS simulator PeP [243] already proposed to use some simple arithmetic predicates to control the applicability of the rules. With GNPS we propose to go further and to use conditional rules of form (a semicolon is used to separate variables located in two sides of a comparison operator):

$$P(x_1, \ldots, x_k; E_1, \ldots, E_m); F(x_1, \ldots, x_k) \to c_1|v_1, \ldots, c_n|v_n, \tag{5.8}$$

where P is predicate in Presburger arithmetic (recall that this is the first-order theory of the natural numbers with addition, i.e. one can use comparisons, Boolean operations, additions, subtractions and constant multiplications in expressions). Moreover, in the basic variant of the definition, production functions F are restricted to be Presburger as well. However, in order to accommodate real-case scenarios, it is allowed the usage of a finite algebraic signature (a set of arithmetic operations and basic functions that can be used in addition to the operations in Presburger arithmetic) for both production functions and predicates.

Example 5.5. *A program catalyzed by a enzymatic variable e in a ENPS has the form*

$$F(x_1, \ldots, x_n)(e \rightarrow)Z$$

In GNPS, this type of prgram is interpreted as

$$P(x_1, \ldots, x_n; e); F(x_1, \ldots, x_n) \rightarrow Z$$

where $P(x_1, \ldots, x_n; e) = e > max(x_1, \ldots, x_n)$.

Example 5.6. *The following is a valid predicate*

$$P(x, y, z; E, F) = E > x \wedge (F > y * 2 + 3 * z).$$

If an algebraic signature contains the ordinary multiplication operation ($\sigma = \{\times\}$), then it would be possible to write the following predicate

$$P(x, y, z; E, F) = E > x \wedge ((F > y * 2 + 3 * z) \vee (E + F > x \times y + z)).$$

Finally, in order to obtain a deterministic evolution of the system, assume that GNPS works in all-parallel mode, i.e. all applicable rules are applied in parallel at each step. This allows to greatly simplify the design of the hardware implementations.

Formally, a GNPS can be defined as the following tuple

$$\Pi = (m, I, O, (Var_1, Var_1(0)), \ldots, (Var_m, Var_m(0)), Pr, \sigma),$$

where m, Var_i and $Var_i(0)$ have the same meaning as in NPS (the number of cells/membranes, the vectors of internal variables and their initial values). The rules are no more specific to some membrane, so they are all collected in the set Pr. Each rule is of form (5.8). In the case of an always true predicate, it can be omitted. Used variables in each rule induce a dependency hypergraph. When this hypergraph is a tree, one can use a Venn diagram notation and place rules in corresponding cells/membranes. The input (resp. output) variables are given by the set I (resp. O). The algebraic signature σ contains the list of additional functions used (with respect to the addition/subtraction and constant multiplication). If $\sigma = \emptyset$ then it may be omitted from the definition.

To show the modeling ability of GNPS, take into account the predator-prey recurrence relation [244, 245] given in Formula 5.9 which can be derived to formula 5.10 and 5.11. Obviously, these two equations are non-linear for the involved function $f(x) = x^2$. This function is approximated by function $g(x) = |x|$, as displayed in Figure 5.8. The reason why approximating nonlinear functions with linear ones is that the computation of linear functions is faster and consuming less hardware resources and powers than that of nonlinear ones, at the expense of lower accuracy. There are four sign combinations of $H[t] + L[t]$ and $H[t] - L[t]$ which correspond

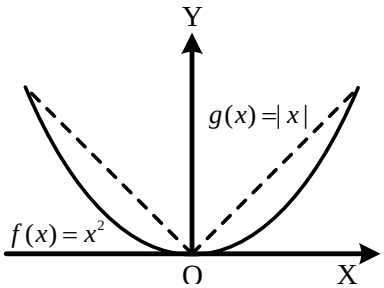


FIGURE 5.8: Linear approximation of $f(x) = x^2$ by $g(x) = |x|$.

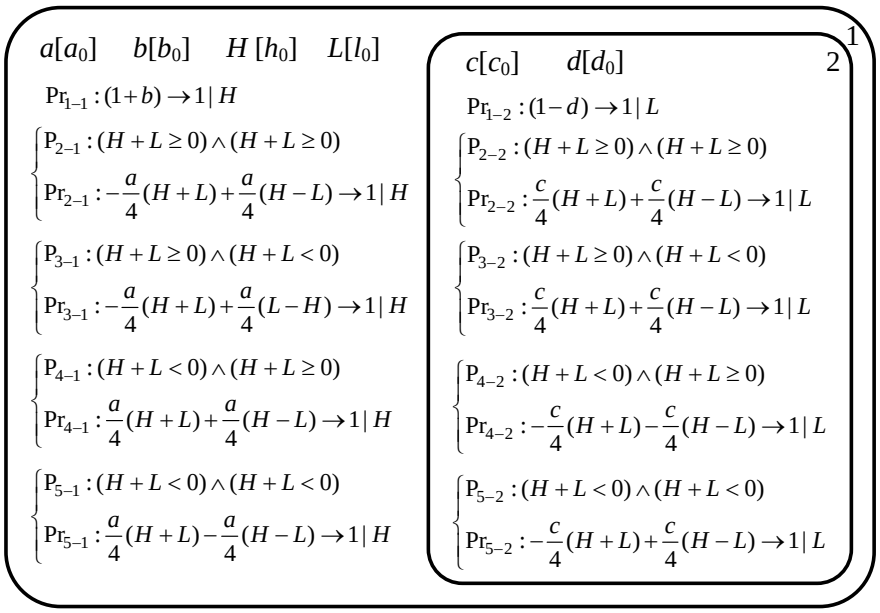


FIGURE 5.9: The predator-prey GNPS. A predicates and its associated rule are placed in two rows in a brace to show them more explicitly.

to four different conditions. So the GNPS performing Formula 5.9 is designed as shown in Figure 5.9.

$$\begin{cases} H[t+1] = H[t] + bH[t] - aH[t]L[t] \\ L[t+1] = L[t] - dL[t] + cH[t]L[t] \end{cases} \tag{5.9}$$

$$\begin{aligned} H[t+1] &= H[t] + bH[t] - aH[t]L[t] \\ &= (1+b)H[t] - \frac{a}{4}(H[t] + L[t])^2 + \frac{a}{4}(H[t] - L[t])^2 \end{aligned} \tag{5.10}$$

$$\begin{aligned} L[t+1] &= L[t] - dL[t] + cH[t]L[t] \\ &= (1-d)L[t] + \frac{c}{4}(H[t] + L[t])^2 - \frac{c}{4}(H[t] - L[t])^2 \end{aligned} \tag{5.11}$$

5.6 FPGA implementation of GNPS

According to the discussion in Section 5.2 and the fact that Presburger arithmetic is recursive, any GNPS system can be rewritten as the following time series (where $X(t)$, $Y(t)$ and $Q(t)$ are the vectors of input, output and internal variables, respectively, at time t):

$$\begin{cases} Q(t+1) & = F(Q(t), X(t)) \\ Y(t+1) & = G(Q(t), X(t)) \end{cases} \tag{5.12}$$

These equations are the generalization (using real numbers instead of Boolean values) of equations used in switching algebra [232] for the definition of the concept of Mealy automaton [195], which together with Moore automaton [200] form the basis of modern synchronous circuit design. Since from the implementation point of view real numbers should be encoded using a fixed bit size, it appears that GNPS are very similar to vector Moore/Mealy machines. This in turn allows a straight implementation using hardware FPGA technology.

For FPGA implementation of GNPS, representations of membrane structures, variables, programs and their executions should be considered firstly. According to discussions in Chapter one, membrane structures are represented implicitly by representations of their inclusions, namely variables and programs for the case of GNPS. Registers are used to represent variables as well. Since FPGA implementations of NPS have not been found in literature, there are no reference methods for representation and parallel execution programs on FPGA. An program implementation method named *FPGA step-wise parallel implementation of program* (FSPIP) is devised. In this method, program applicability predicates are expressed as logical expressions in the HDL *if* condition statement. Production functions of program are generally algebraic expressions which can be characterized by HDL arithmetic operators conveniently.

Re-partition protocols allocate values of value production functions to relevant variables proportionally by virtue of division operations triggered after the completion of computing of all applicable production functions. Multiple re-partition protocols can assign values to the same variable. However, updating variables with multiple values at the same time will cause conflicts. To address this problem, all partitioned values of a variable are summed first, and then assigned to the variable to avoid conflicts. All production functions of applicable programs are triggered simultaneously on the rising edge of clock so that the parallelism is realized. Then delay an appropriate amount of time (usually one clock cycle), firing divisions of multiple repartition protocols. The partitioned values of variable are then added together and assigned to the corresponding variables to complete executions of applicable programs. The main procedures of the FSPIP method are shown in Figure 5.10. It should be noted that the FSPIP method is only practicable for GNPS with fixed-point variables. For GNPS using floating point variables, a new method is contrived to implement programs, as detailed in Chapter 7.

For implementation efficiency, consider the following restrictions for GNPS at first:

- Real values are replaced by their approximation using a fixed-point binary representation.
- The production functions are linear.
- The predicates are Presburger-definable.

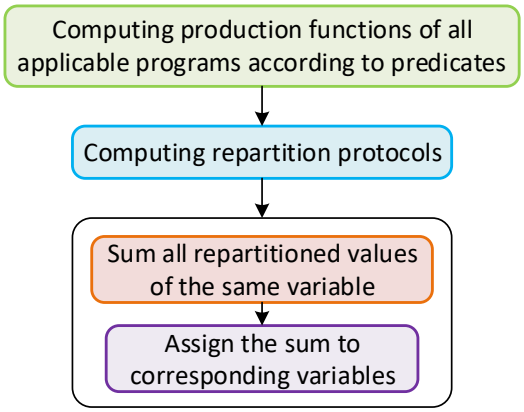


FIGURE 5.10: Main procedures of FSPIP method.

The above restrictions allow to relatively easy obtain the Mealy/Moore machine in form of Equation (5.12). The corresponding functions F and G are linear enriched with conditional statements as it is shown in Example 5.7.

Remark 5.1. In the basic case just consider an empty signature σ as this allows a straight translation to Verilog. For more complex computations, corresponding functions should be implemented additionally as Verilog modules. This can induce delays, as in many cases it is not possible to compute corresponding functions in one clock step.

Remark 5.2. In the case of fixed-point encoding, it is possible to easily implement the multiplication operation working in one time step. This can be done either directly (by using multiplication code dependent on the width of the encoding), or using a special component of FPGA called *DSP slice* that allows to perform multiplication operations in one step (up to 48-bit width).

Remark 5.3. Contrary to multiplication, it is not easy to implement the division operation in one time step. However, the division by a constant c can be seen as the multiplication by c^{-1}.

Example 5.7. *Consider the system depicted on Fig. 5.11. It has two input, two output and two internal variables. The system computes the average value of its inputs and also indicates if this value changed by more than 0.1% on the previous step. Recall that all rules are executed in parallel.*

The set of equations corresponding to this system is the following ($a(0) = b(0) = out_1(0) = out_2(0) = 0$).

$$\begin{aligned} a(t+1) &= \frac{in_1(t) + in_2(t)}{2} \\ b(t+1) &= a(t) \\ out_1(t+1) &= a(t) \\ out_2(t+1) &= \text{if } |a(t) - b(t)| < 0.001 \text{ then } 0 \text{ else } 1 \end{aligned}$$

It can be directly transformed to Verilog as follows (assume a fixed point encoding of real numbers over 32 bits and using 12 bits for the fractional part). In the below listing the fixed-point (constant) multiplication is performed by the function `_mult` *(recall that* 2048 *is* 0.5 *in the chosen fixed-point encoding).*

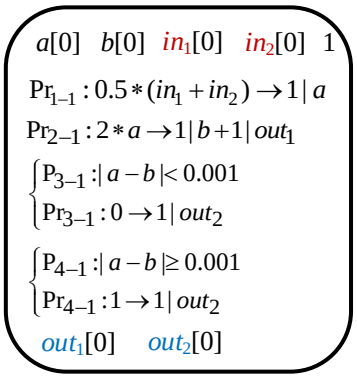


FIGURE 5.11: GNPS system from Example 5.7. The predicate for each program is taken to a separate line before it. Variables in red and blue indicate input and output variables respectively. Others are intermediate variables.

```
1  module A #(parameter WIDTH = 32, parameter BPPOS = 12)
2  (    output [WIDTH:0] out1, output [WIDTH:0] out2,
3       input  [WIDTH:0] in1, input  [WIDTH:0] in2,
4       input clk
5  );
6       reg [WIDTH:0] a = 0;
7       reg [WIDTH:0] b = 0;
8       reg [WIDTH:0] c = 0;
9       reg [WIDTH:0] d = 0;
10
11 always @(posedge clk) begin
12      a <= _mult(in1+in2, 2048); //0.5
13      b <= a;
14      out1 <= a;
15      out2 <= a-b < 4 && a-b > -4 ? 0 : 4096;//0.001, 1
16 end
17 endmodule
```

It can be seen that the translation is rather straightforward. A compiler `FPNtoVerilog` was developed in order to assist in this translation. As input it takes the GNPS model in form of Equations (5.12) and produces as output behavioral Verilog code implementing the corresponding Mealy/Moore automaton.

The compiler performs the following steps:

1. Parse the input file.
2. Identify input and output symbols.
3. Flatten the obtained system.
4. Perform constant propagation.
5. Convert all constants to fixed-point real number representation.
6. Write Verilog output.

These steps are performed using standard compiling techniques. The last step is straightforward as a sequential switching function/circuit can be directly translated to Verilog. As a result a file containing the synthesizable (in FPGA) Verilog module whose code simulates each step of the GNPS at each clock cycle is generated. Two case studies are designed as target models and their FPGA implementation process

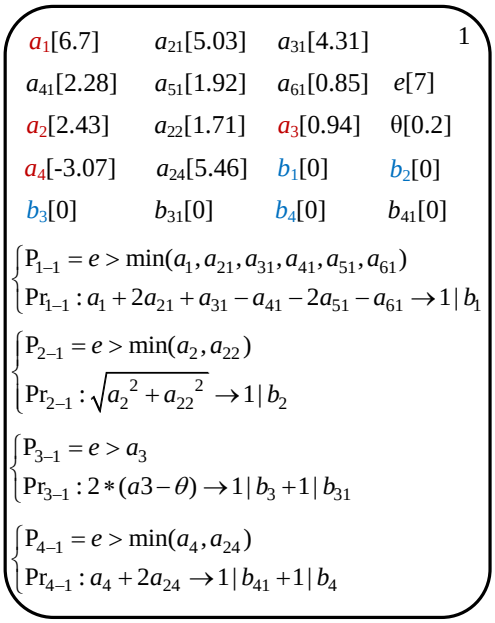


FIGURE 5.12: GNPS model for case study 1. It implements the core computations of Sobel image edge detection algorithm. The predicate P_i and program Pr_i are written in two lines to render them better. GNPS1 has a skin membrane containing 4 programs and 19 real-value variables. Input variables are highlighted in red. Output variables are highlighted in blue.

is detailed to elucidate how a GNPS can be implemented in a FPGA. Since the algorithm implemented is making use of the square root function, it is considered that the signature of the system is $\sigma = \{\sqrt{\cdot}\}$.

The target development board is Digilent BASYS 3 equipped with a Xilinx Artix-7 xc7a35t-1cpg236c FPGA as core component. The FPGA developing environment is Xilinx Vivado 2019.1 and Verilog is used as HDL. A Dell *Latitude* outfitted with a Intel Core i7-7820HQ and 16 GB RAM is the host computer.

5.7 Case studies

5.7.1 Case study 1

The GNPS model (called GNPS1 for simplicity) of case study 1 is illustrated in Figure 5.12. It stems from Sobel image edge detection algorithm. GNPS1 only has a skin membrane, without any inner membranes. A program is applicable if its conditional rule can be met. Variable e is assigned a big enough value so that the 4 programs can be executed at the same time. Fixed-point number format is employed to represent real values. Specifically, every variable is assigned a 20-bit register in which the first bit designates the sign bit, the following 8-bit denoting integer part and the rest of 11-bit presenting fraction part of a real number. The signature $\sigma = \{x^2, \sqrt{\cdot}\}$ is also used.

After inputting GNPS1 to the `FPNtoVerilog` compiler, the output is a behavioral model that specifies the behavior of GNPS1. Implementations of the signature σ (square root and the square function) is also provided. Next, this model is translated to register transfer level (RTL) using Vivado tools. The upper-most level of

GNPS1 schematic generated by Vivado is depicted in Figure B.1. Then verify this model in a test bench. In this research, two case study models are constructed as sequential circuits [246], namely clock is involved as a metronome to synchronize operations. Rising edge of clock is the trigger signal, i.e., operations can only be carried out after a rising edge and variables hold their values until next rising edge. The period of clock is set to 10 ns in the test bench.

The behavioral simulation conducted by the test bench omits any gate delays and data path delays, which means that results are output instantaneously, at the same time of trigger edge for sequential circuits and the changing instant of signals for combinatorial circuits. Corresponding Vivado IP core [247] is resorted to compute the square root. Post implementation timing simulation which can only be conducted after *implementation* operation can provide more reliable timing waveform.

Models can be synthesized after behavioral simulation if it behaves as expected behaviors. There are two important tasks should be done after synthesizing target model: setting constraints and debug cores. Constraints include timing constraints and physical constraints. In timing constraints, the period of the clock and input/output delays are set, while physical constraints specify I/O configurations, mapping ports of model to pins of FPGA. The clock period is set 10 ns in the two case studies. To save pins, only b_3 and b_4 are set as output ports for GNPS1. All the constraints are written in constraint file (.xdc) in the format of industry standard Synopsys Design Constraints (SDC) [132]. The variables to be debugged in hardware debug procedure are marked and set after synthesis.

The subsequent procedure of *Synthesis* is *Implementation*, which performs place & route of the synthesized circuits and other vital operations such as route optimization, power and resource consumption analysis and timing analysis. Gate delays and data path delays of a model are taken into account after implementation so the timing of a design can be revealed by post implementation timing simulation. Variable b_2 obtains its stable value during 155$\sim$160 ns, while other variables get their stable values more early, as shown in Figure 5.13. According to design timing summary provide by Vivado, the worst negative slack (WNS) is 3.97 ns, worst hold slack (WHS) 0.058 ns and worst pulse width slack (WPWS) 3.75 ns. These values are positive so timing requirements are met.

For the sake of comparing the computing speed of FPGA hardened GNPS and software simulation of GNPS, speedup is defined as the ratio of elapsed time of two methods. A software called *PeP* which dose not have a GUI dedicates to emulate (E)NPS [243]. To simulate other types P systems, one can resort to P-Lingua [248]. GNPS1 is transformed to its ENPS counterpart and emulated by *PeP*, which outputs results and elapsed time to compute the results, as shown in Figure 5.14. PeP costs 0.005651 s to finish the simulation. Then the speedup of FPGA hardened GNPS1 is calculated in Equation 5.13. The maximum error of output variables is computed in Equation 5.14. In fact, results of *PeP* round to nearest and only save two significant digits behind decimal point. If we do the same for the FPGA result, there would be no discrepancy.

$$\frac{5.651 \times 10^6}{160} = 3.5319 \times 10^4 \tag{5.13}$$

$$\left| \frac{3.92 - 3.92480469}{3.92} \right| \times 100\% \approx 1.225765 \times 10^{-3} \tag{5.14}$$

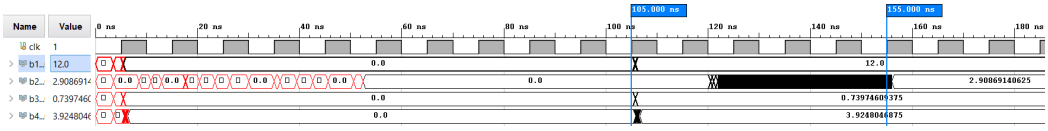


FIGURE 5.13: Post implementation timing simulation of GNPS1. Port b_1, b_3 and b_4 obtain their steady output value after the eleventh rising edge of clock, indicating it costs 110 ns to get results. For b_2, its steady output value emerges after sixteenth rising edge, costing 160 ns to compute outcome.

```
INFO:Simulation finished succesfully after 1 steps and 0.005651 seconds; End state below:
num_ps = {
  Skin:
    var = { a1: 0.00,  a21: 0.00,  a31: 0.00,  a41: 0.00,  a51: 0.00,  a61: 0.00,  a2: 0.00,  a22: 0.00,  a3: 0.00,  a4:
 0.00,  a24: 0.00,  b1: 14.10,  b2: 2.97,  b3: 0.74,  b31: 0.74,  b4: 3.92,  b41: 3.92,  theta: 0.00, }
    E = { e: 7.00, }
}
```

FIGURE 5.14: Software simulation of GNPS1. It is assumed that GNPS1 evolves one step to stop. There is no one-to-one correspondence between a clock cycle and a GNPS step. For complex arithmetic computations, one step of GNPS requires more than one clock cycle.

The estimated resource dissipation and power consumption of GNPS1 is reported after implementation, shown in Figure B.2. Because the function of GNPS1 is not complicate, the dynamic power merely shares 13% to 14% of total power and clock power makes up more than 70% of dynamic power.

The real computing results of FPGA cannot be observed straightforwardly, but requires a particular procedure called *hardware debug*. Variables to be debugged should be marked in Verilog codes or marked in the net list. b_3 and b_4 are marked in Verilog codes as debug signals. After programing device, the integrated logic analyzer (ILA) window opens automatically. Debug signals should be added into the window manually, then run debug to exhibit values computed by FPGA, shown in Figure 5.15.

From Figure 5.13 we can see that b_1,b_3,b_4 get their results almost at the same time, while b_2 obtains its result 50 ns later. The reason behind this latency is that the program calculating b_2 involves square and square root which are more time-consuming than normal arithmetic operations computing b_1,b_3 and b_4. So FPGA hardened GNPS1 behaves as expected to carry out parallel computing and to get right answers comparing to results software simulation.

5.7.2 Case study 2

In practical applications such as image processing and robot path planning, computation process comprises several sequential procedures. In each procedure, multiple functions may be performed in all parallel mode, like the way GPNS1 works. In this subsection, GNPS1 is modify from all parallel to sequential mode, resulting GNPS2, depicted in Figure 5.16. Conditional rule of membrane 2 is met at beginning, so program Pr_{1-1} and Pr_{2-1} take place simultaneously. Other conditional rules are not met for the initial value of conditional variables (e_i) are zeros. After Pr_{2-1} modifying the value of e_2 from 0 to 3, programs in membrane 3 are triggered to execute. So does membrane 4 and 5. It is worth to note that Pr_{2-2} and Pr_{2-3} consume e_1 and e_2 so

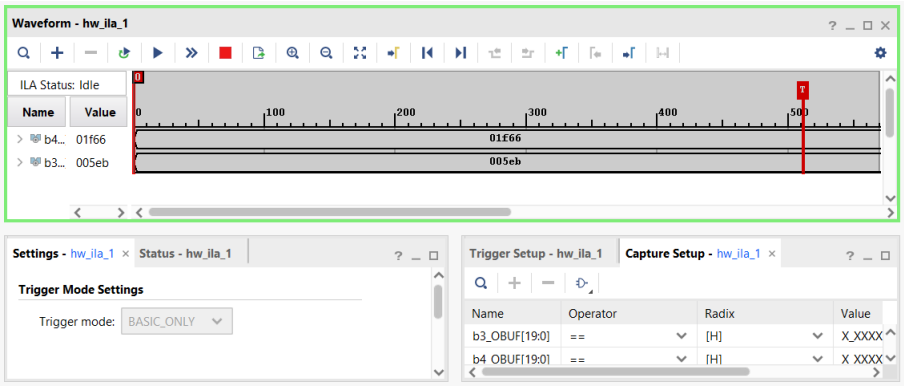


FIGURE 5.15: Hardware debug of GNPS1. Input variables cannot be debugged so there is no clock signal. Values are represented in hexadecimal, 01f66 is 8038 in decimal. $8038 \div 2^{11} = 3.9248046875$, which is the value of b_4. 005eb is 1515 in decimal, $1515 \div 2^{11} = 0.73974609375$, which is the value of b_3.

each program can only execute once. In short, a sequential ignition method is used to control the execution sequence of programs.

Input GNPS2 to `FPNtoVerilog` obtaining the RTL model. The block diagram of GNPS2 RTL model is illustrated in Figure B.3. Edge detection is used as the trigger signal to impel the next membrane to execute. By this way, the train-like ignition is realized. Synthesize GNPS2 and open the synthesized model, complete constraints design and debug core set as that of GNPS1, then implement GNPS2. Run post implementation timing simulation to check the timing situation, as shown in Figure 5.17. *PeP* simulation of GNPS2 shown in Figure 5.18, PeP takes 0.009306 s so the speedup of FPGA implementation is computed in Equation 5.15.At last, perform hardware debug to verify that FPGA hardened GNPS2 obtained correct outcomes, shown in Figure 5.19.

$$\frac{9.306 \times 10^6}{480} \approx 1.9388 \times 10^4 \tag{5.15}$$

Figure 5.17 indicates that the four output variables are computed in sequential as designed. Again, the calculation of b_2 costs the longest time for its complexity. Finally, the last tests were performed using an autonomous execution of the system without output and using distributed read-only memory data storage for the input. Under this setup the speed of 100Mhz was achieved. This means that a GNPS model can be simulated at a speed of 10^8 steps per second. It is remarked that in real-use cases the reaction speed will be dependent on the input/output delay. The input/output circuits are not system-specific and can be reused in different situations. However, at the present state they need to be integrated manually in the final hardware design. So the development of `FPNtoVerilog` continues in order to integrate the automatic generation of input/output modules. This will allow a generation of a hardware circuit directly from the GNPS specification, without any user intervention.

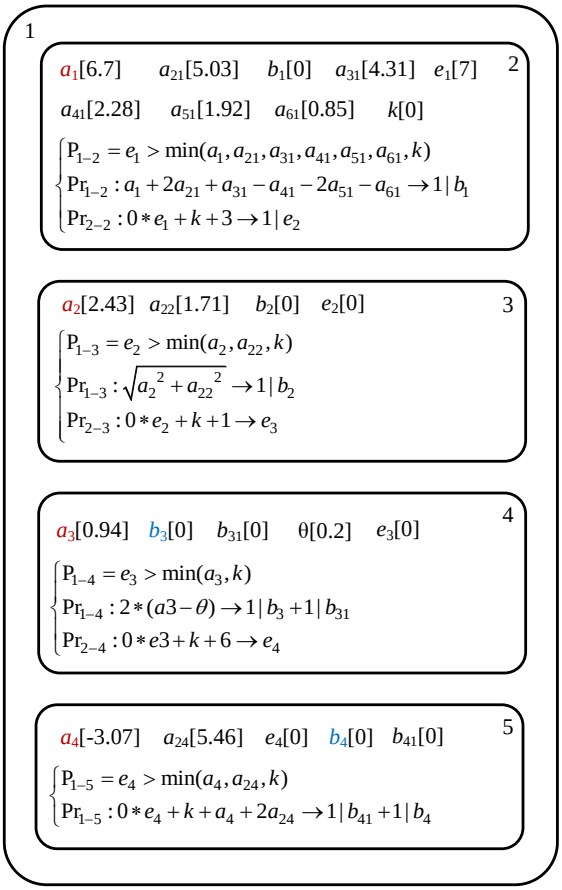
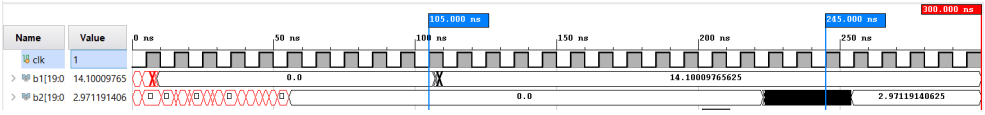
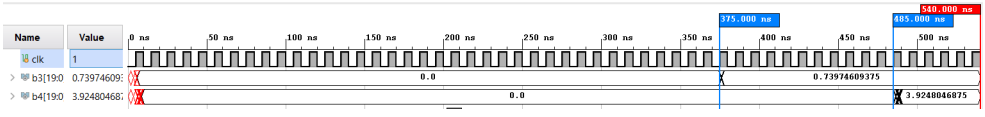


FIGURE 5.16: GNPS model for case study 2 is numbered as GNPS2. The equations inside are the core computations of Sobel image edge detection algorithm. GNPS2 has 5 membranes and evolves 4 steps to reach halt condition. Programs in each membrane compute concurrently while each membrane execute serially.



(a) Variable b_1 gets its value in the 11th cycle, while b_2 obtains its value in the 25th cycle.

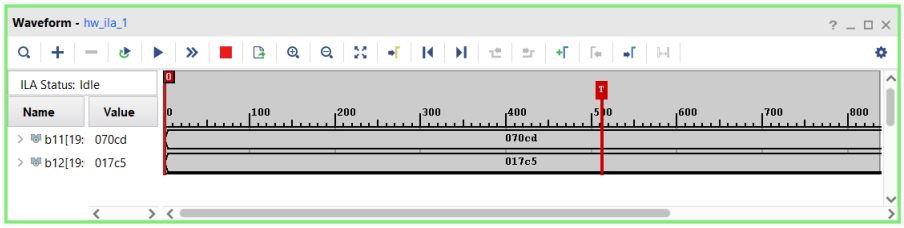


(b) Variable b_3 gets its value in the 38th cycle, while b_4 obtains its value in the 49th cycle.

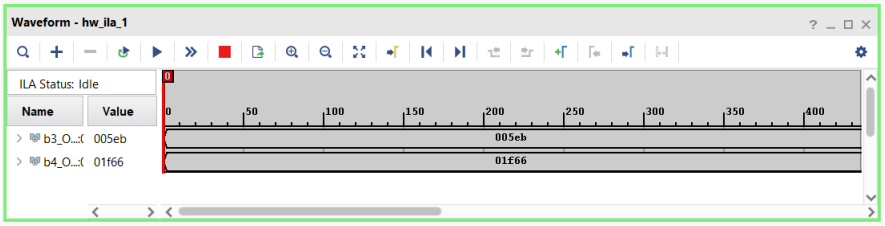
FIGURE 5.17: Post implementation timing simulation of GNPS2. The real timing of b_2 and b_3 is a little different than expected.

```
INFO:Simulation finished succesfully after 4 steps and 0.009306 seconds; End state below:
num_ps = {
  Skin:
    var = {}
    E = {}
  Mem1:
    var = { a1: 0.00,  a21: 0.00,  a31: 0.00,  a41: 0.00,  a51: 0.00,  a61: 0.00,  k: 0.00,  b1: 14.10, }
    E = { e1: 0.00, }
  Mem2:
    var = { a2: 0.00,  a22: 0.00,  b2: 2.97, }
    E = { e2: 0.00, }
  Mem3:
    var = { a3: 0.00,  b3: 0.74,  b31: 0.74, }
    E = { e3: 0.00, }
  Mem4:
    var = { a4: 0.00,  a24: 0.00,  b4: 3.92,  b41: 3.92, }
    E = { e4: 0.00, }
}
```

FIGURE 5.18: Software simulation of GNPS2. It is assumed that GNPS2 evolves four steps to stop. CPU of the host computer costs 0.009306 s to get results.



(a) Hardware debug of b_1 (connect to b_{11}) and b_2 (connect to b_{12}).



(b) Hardware debug of b_3 and b_4.

FIGURE 5.19: Hardware debug of GNPS2. Values are represented in hexadecimal, 070cd is 28877 in decimal. $28877 \div 2^{11} = 14.10009765625$, which is the value of b_1. 17c5 is 6085 in decimal, $6085 \div 2^{11} = 2.97119140625$, which is the value of b_2.

5.8 Wrap-up

Definitions of NPS and ENPS are given at first. The relation of (E)NPS and system of difference equations are analyzed: (E)NPSs are another forms of difference systems. Considering the relationship between difference systems and differential systems, (E)NPS can model application models characterized by differential systems. This result shows that (E)NPS has good prospect in engineering applications for their large scale parallelism which can speedup computations. Binary and unary form of (E)NPS is came up with to facilitate FPGA implementation. The source of different forms between symbol-object cell-like P system and NPS have dug some how. And to go a step further towards this goal, the notion of GNPS is proposed. A compiler from difference system to GNPS is devised to generate Verilog codes of RTL model. Two case studies show that FPGA implementation of GNPS can be done in a straightforward manner with a speedup of 10^4 for the algorithm handled.

Chapter 6

FPGA implementation of robot membrane controller

As a new branch of nature computing, the research of membrane computing practical application can not keep pace with its fruitful achievements in theoretical aspect [249]. This situation was taken seriously by membrane community and some scholars have engaged in applications of P systems ever since a long time ago. Keeping in mind the biological background of membrane computing, using P systems as modeling framework for biological processes and ecosystems were the early explorations to apply membrane computing, referring [56, 57, 59, 61]. By resorting to P systems as modeling framework, Giant panda population dynamics modeling is another target under research.

The non-determinism of P systems is a valuable nature for biological and ecological system modeling. Nevertheless, for engineering applications, non-determinism is the property trying to avert. Beginning from 2015, several variants of fuzzy spiking neural P systems (FSNP) have been used in power system fault diagnosis [250] [251] [157] [252] [253] [254], setting a new direction for P system applications. By setting one rule per membrane, the non-determinism of FSNP is removed. Complex market interactions are modeled by population dynamic P systems in [255]. Other applications can be found in [256] [257] [258]. The large scale parallelism of P systems turned out to be quite favorable for real life applications.

Since 2011, adopting NPS and ENPS to model autonomous mobile robots controller has been another research highlight of P system applications. Membrane controllers designed in [238] were NPS systems designed to act as controllers and running in some environment. As the first case putting NPS to real life application, in [238], three NPSs are developed as the controllers for Khepera III and e-puck robots to perform obstacle avoidance, wall following and following leader behaviors. These three NPSs are simulated by a software called SNUPS, which is designed as Java servlet. When running the robot, it invokes the SNUPS engine. After computing, the results will be returned to robot to control motors' speeds, performing specific behaviors. Both experiments on simulated robots and real robots were conducted to verify the control effect of NPS. In [259] a kinematic controller and a proportional-integral-derivative controller based on ENPS are designed for wheeled mobile robots. Another interesting application is found in [260] where an environment classifier and a novel multi-behaviors control approach are proposed to enhance the reactive navigation performance of autonomous mobile robots.

Membrane controllers require several ingredients. The motion of a robot in a real or simulated environment requires a program that reads/transmits the values of robot' sensors (usually distance and speed), runs the simulation of the controller for one or several steps and then updates/transmits the values of actuators (usually

robot wheel motors). Before running the simulation this program should assign initial values for the membrane controller and after the simulation it should retrieve corresponding output values from it. The controller itself is simulated using a custom simulator [238, 261] or by Matlab code [259, 260]. In some cases [261, 260] the experiments were carried out in real robot environments. To speed-up the simulation in the case of complex controllers [262] the use of graphical processing units (GPU) hardware architectures was proposed [263].

The first ENPS robot controller performing obstacle avoidance behavior was proposed in [264]. It is a general controller not targeting particular robots. While no experiments were conducted by the ENPS software simulator eSNUPS [218], which is an extended version of SNUPS. The portability of NPS and ENPS robot controllers was validated in [265] by adapting the control law and the number and placement of sensors, and the dimension parameters of robots. ENPSs with different functionality were developed later on, expanding the utilizing range of ENPS besides robot motion control. For instance, an ENPS do the robot localization was presented in [239]. Robot trajectory tracking ENPS was designed in [259]. Particle swarm optimization based robot path planning ENPS was introduced in [266]. ENPSs performing image edge detecting were proposed and validated in GPU [267]. This chapter focuses on FPGA implementation of robot membranes built on NPS, ENPS and GNPS. The FPGA implementation of GNPS is compatible for NPS and ENPS because GNPS is a superset of them.

6.1 FPGA implementation of membrane controller based on NPS

The first obstacle avoidance NPS controller was proposed in [238], targeting e-puck robot, which equips 8 infrared sensors around the body. Equation 6.1 is the obstacle avoidance control law built in the framework of a NPS in which *lw* and *rw* are the speed value accepted by left and right motor as the required speed to follow. This control law is named as control law 1 hereinafter. *CruiseSpeedLeft* and *CruiseSpeedRight* are the cruise speed of both wheels when no obstacles are detected. x_i is the original readings of sensors telling the distances (positively correlated, expound in the next paragraph) between robot and obstacles while s_i is the transformed sensor reading negatively correlated to obstacle distances. M is a large positive number having value of 1000. $weihtLeft_i$ and $weihtRight_i$ are the weight values of sensors placed in the left and right hand side of robot. Variables in the second equation are the counterparts of those in first one. This control law is to be applied on a bigger robot–Pioneer 3 DX which is a product of Omron Adept Mobile Robots LLC. Pioneer 3 DX is outfitted with 16 sonar sensors arranged in 2 arrays, whose placements are shown in Figure 6.1. i takes values from 0 to 15, corresponding to the numbers of sensors.

$$\begin{cases} lw = CruiseSpeedLeft + \sum_{i=1}^{16} s_i * weihtLeft_i \\ rw = CruiseSpeedRight + \sum_{i=1}^{16} s_i * weihtRight_i \\ s_i = -x_i + M \end{cases} \tag{6.1}$$

Both the infrared sensors in e-puck and sonar sensors in Pioneer 3 DX return the distances between the robot and obstacles. But the property of Pioneer 3 DX sonar sensors make the adaption of control law not so straightforwardly. Specifically, the difference of two distances sampled by these 2 robots lays in that e-puck's

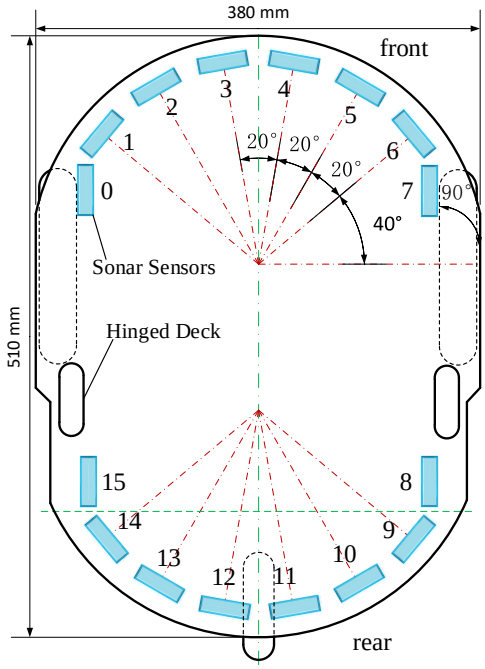


FIGURE 6.1: This is a plan view of Pioneer 3 DX robot which is covered by a hinged deck on the top. The 16 rectangles in light blue are the sonar sensors surrounding the robot, just beneath the hinged deck. Sensors are arranged in two arrays in the front and in the rear. The layout of sensors in two arrays is identical.

infrared sensors return value 0 when there are no obstacles and the reading of sensors increase as the distances decrease. It is called that the reading and distance are negatively correlated. While Pioneer 3 DX's sonar sensors return the distances of the robot to obstacles, the reading of sensors decrease as the distances decrease so it is said positively correlated. The transformation given by the last equation of Formula 6.1 was done to the readings of Pioneer 3 DX sensors so that the control law can be adopted. The NPS used as robot controller is illustrated in Figure 6.2 which computes three steps to output results (speeds of left and right wheel).

In accordance with general ideas of FPGA implementation of P systems, membranes are represented implicitly by synchronizing the execution of rules/programs in different membranes. Different rules/programs will be mapped to different hardware resources in FPGA so that the distributive nature of P systems is achieved without membrane structures.

In Figure 6.2, variables $weightLeft_i$ and $weightRight_i$ $(i = 1, \ldots, 16)$ are sensors' weights whose values reflect the influences of sensors on different positions to the speed of left and right wheel. Supposing a obstacle is detected on the left side, the speed of left wheel should be larger than the speed of right wheel so that robot can turn right to avoid this obstacle. Based on this assumption, the weight values of sensor 0, 1, 2, 3, 15 should impose negative effects to right wheel speed in order that right wheel speed is diminished. On the contrary, sensor 0, 1, 2, 3, 15 should impose positive effects to left wheel speed in order to raise its speed. Similarly, sensor 4, 5, 6, 7, 8 impose negative effect to left wheel but positive effect to right wheel. Sensors located in the rear part of the robot are unhelpful for detecting obstacles in front, so their weight values are set to zero.

Consequently, the weight values of variables $weightLeft_i$ and $weightRight_i$ have inverse values to manifest the positive and negative effects. Pioneer 3 DX should

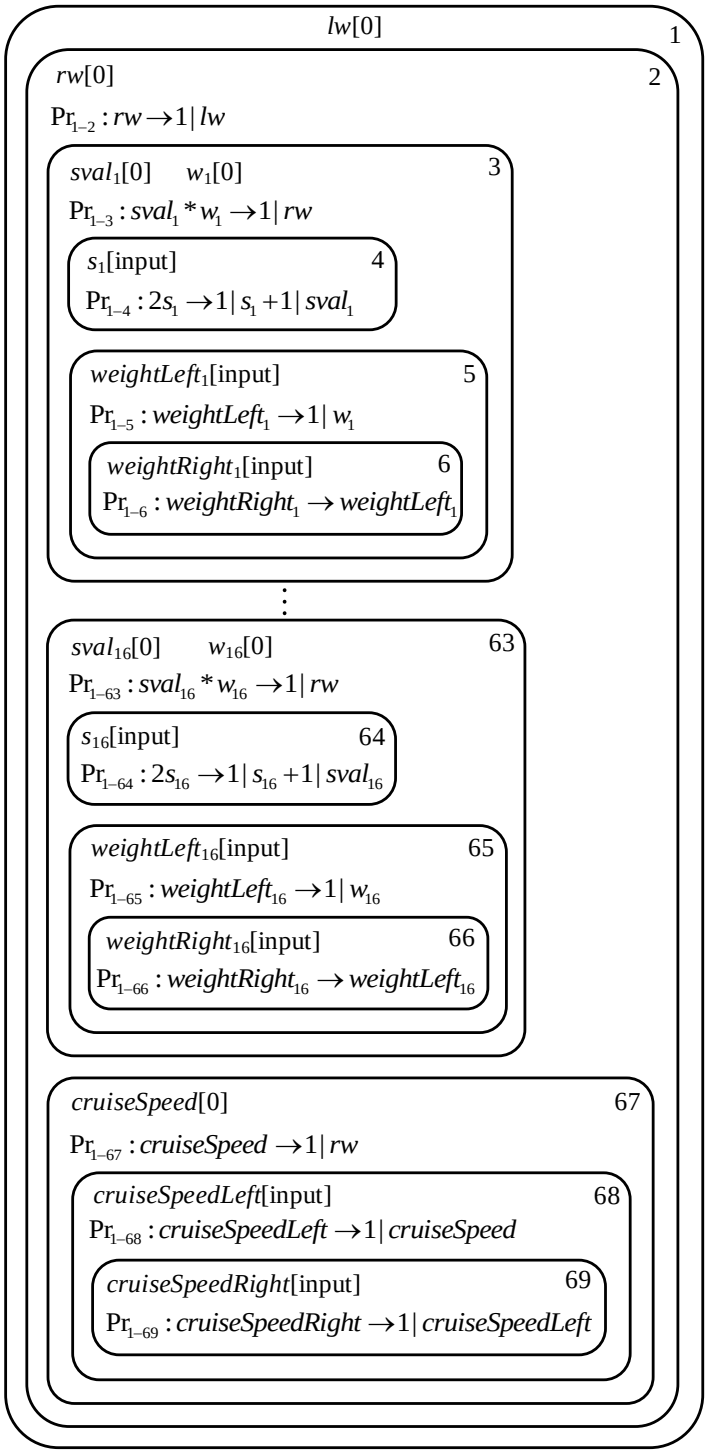


FIGURE 6.2: The NPS controller performing control law of Formula 6.1, which will be implemented in FPGA. This numerical P system is called NPS1 below.

TABLE 6.1: The calibrated values of $weightLeft_i$ and $weightRight_i$, along with a set of sampled sensors reading data which will be utilized to verify the correctness of RTL model of NPS1.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
wl_i	0.1	0.4	0.6	0.8	-0.8	-0.6	-0.4	-0.1	-0.1	0	0	0	0	0	0	0.1
wr_i	-0.1	-0.4	-0.6	-0.8	0.8	0.6	0.4	0.1	0.1	0	0	0	0	0	0	-0.1
s_i	277	0	0	0	0	0	17	208	190	576	704	745	733	659	451	296

be calibrated to determine the values of $weightLeft_i$ and $weightRight_i$ firstly. The general calibration process can be stated as follows: use control law given in Formula 6.1 to control robot, assigning some initial value to $weightLeft_i$ and $weightRight_i$ then running Pioneer 3 DX. If it bumps into obstacles, altering initial values of these two arrays to some extent in line with the collision severity until it no longer rams any obstacles. The corresponding calibrated values of these two variable arrays are shown in Table 6.1.

Membranes of NPS1 can be classified into two types according to their functions: delivery membranes and computing membranes. Membrane 4∼64, 5∼65, 6∼66, membrane 2, 68 and 69 are delivery membranes whose duties are transmitting value of variable to another. For instance, values of variables $weightRight_i$ are sent to variables $weightLeft_i$, which are transferred to variables w_i furthermore. Membranes 3∼63 and 67 are computing membranes to calculate new value of variable *rw*. Computations performed by computing membranes should be synchronized to reflect the parallelism of NPS. Keep in mind that NPS1 should compute three steps to get results in such a cycle: after the first step finished, $rw = 0$, $lw = 0$; for the second step, *rw* obtained the left wheel speed value which will be assigned to *lw* in step 3 and $lw = 0$; in the next step *rw* acquires the expected right wheel speed value and *lw* attains the second step value of *rw*. This process repeats if computing proceeds.

To coordinate the value transfer process in delivery membranes, counters are adopted aiming at this action. Taking the value transfer in membrane 5∼65 and 6∼66 as an example, the initial values of variables $weightRight_i$, $weightLeft_i$ and *counter* are zeros. *counter* counts in a loop from 0 to 2, to correspond computing step 1 to step 3. At the first rising edge of clock, assign the values in Table 6.1 to $weightRight_i$ and $weightLeft_i$. At the second rising edge of clock, the values of $weightRight_i$ are back to zeros and keep these values until the end of the third clock cycle since their values are consumed by production functions and they do not appear in any repartition protocols. The values of $weightRight_i$ loop in accordance with *counter* value loop. However $weightRight_i$ should transfer their values to $weightLeft_i$ in the second cycle. During a loop of *counter*, $weightRight_i$ equal zero from the second cycle, so $weightLeft_i$ also have values zeros in the third cycle. Whether this arrangement is correct or not can be deduced from timing waveform of variables, which will be detailed as follows.

Membranes 4∼64 are omitted in RTL model by substituting programs $sval_i * w_i \rightarrow 1|rw$ with $s_i * weightLeft_i \rightarrow 1|rw$. Because the effect of program $2s_i \rightarrow 1|s_i + sval_i$ is to assign sensors' readings to $sval_i$ and program $weightLeft_i \rightarrow 1|w_i$ in membrane 5∼65 is to transfer the valus of $weightLeft_i$ to w_i, executing programs in membranes 4∼64 will cost 1 more clock cycle which can be reduced by performing $s_i * weightLeft_i \rightarrow 1|rw$. Assuming that variables s_i have the sonar sensors' readings as initial values, computing membranes (membranes 3∼63 and 67) are triggered to compute *rw* at rising edge of clock. In the first clock cycle, $rw = 0$ for the initial values of $weightLeft_i$ are zeros. In the second cycle, *rw* obtains the speed of left

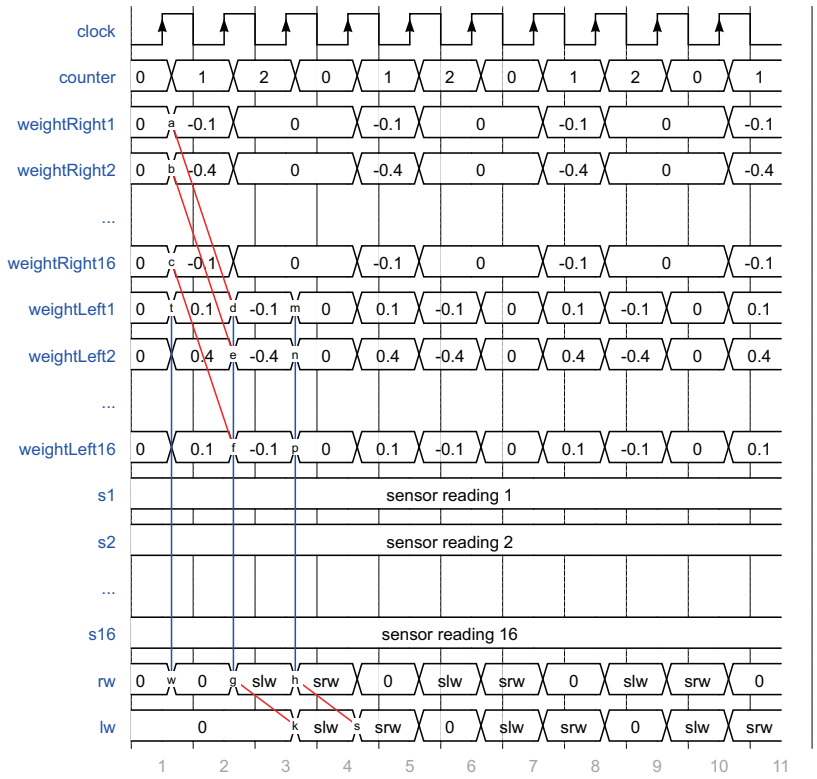


FIGURE 6.3: Expected timing waveform of NPS1 RTL model. Red lines indicate value transfers from $weightRight_i$ to $weightLeft_i$ and from *rw* to *lw*. Blue lines signifies parallel computing of programs in associated membranes.

wheel because $weightLeft_i$ got their exact values (so do $weightRight_i$) during second cycle. In the third cycle, *rw* acquires the speed of right wheel because $weightLeft_i$ got the values of $weightRight_i$ which are transferred at the beginning of the third cycle. Hence this arrangement of transferring and computing can achieve the computation procedures of NPS1 accurately which computes 3 steps to get results. The whole processes of transfer and computing should perform as the timing diagram depicted in Figure 6.3.

Two types of Verilog HDL modules are designed to carry out value transfer and computing operations described above. Module **WeightRight** assigns right weight values (in Table 6.1) of sensors to variables $weightRight_i$, while **WeightLeft** assigns left weight values of sensors and transfers the values of $weightRight_i$ to $weightLeft_i$. Analogously, module **CruiseSpeedRight** and **CruiseSpeedRight** do the same thing to variables *cruiseSpeedRight* and *cruiseSpeedLeft*. **CruiseSpeed** transfers the value of *cruiseSpeedLeft* to *cruiseSpeed*. Module **Compute** is designed to conduct parallel computations originated from membrane 3∼63. **SpeedLeft** passes the value of *rw* to *lw*. No module corresponds to membrane 1 for there are no programs within it. Modules are connected to be an entirety according to signals' input-output relationships. For example, the output of **WeightRight** is the input of **WeightLeft**, so connect the output ports of the former one to the input ports of the latter one.

From Figure 6.2, it is obvious that membranes are organized in nested structure. Nevertheless, modules in RTL model are not nested but are independent with each other. Another notable feature of the RTL model is that the function of modules does not conform to the function of membranes. There are no rigorous correspondence between membranes and modules. In spite of these differences, the behaviors of RTL

model and NPS1 are identical: at each computation step, the value of each variable and computing outcome are the same. The identity of computation steps between NPS1 and its RTL model reflects the validity and rationality of the RLT model. The RLT model composed of modules is shown in Figure 6.4 in which the input-output ports and interconnections of modules are clarified.

As can be seen in Table 6.1, the value of variables are real numbers. Unfortunately, real numbers cannot be represented in digital circuits directly. In fact, real numbers are represented in some forms of integers–fixed point number or float point number. In this research, real numbers are transformed to fixed point numbers which are easy to deal with. To be specific, each variable is assigned a 24-bit register. Allocate the first 11 bits to integer part and the following 13 bits to fraction part of a variable value. This bits' manipulation creates a range of $[-(2^{11}-1),(2^{10}-1)]$ ($[-2047,1023]$ in decimal) which includes the value range of sensors' reading $[0,1000]$. Each real number should be transformed to fixed point number before running the RTL model so that results obtained are also in fixed point representation. Consequently, an inverse transformation is necessary to get decimal results.

NPS and ENPS can be simulated by a software named *PeP* [243]. Software simulation results of NPS1 are the benchmarks of its hardware implementation which provides a fair reference. *PeP* can also offer elapsed time (in seconds) to compute some predefined steps. From a hardware point of view, the software simulation is a CPU implementation of an algorithm. So this returned time reflects the performance of the CPU in host computer. Further more, this CPU implementation time is indispensable to compute the speedup of FPGA implementation of NPS. As can be seen in Figure 6.5 which shows the Vivado behavioral simulation waveform, in the first three cycles, *rw-out* which corresponds *rw* holds value 0, 310.6953125, 289.3046875, behaving exactly as expected. Clock cycle is set to 10 nanoseconds in this behavioral simulation.

The host computer is a Dell *Latitude* equipped with a Intel Core i7-7820HQ and 16 GB RAM. Target FPGA of this research is Xilinx Artix-7 xc7a35t-1cpg236c which is the core part of BASYS 3 FPGA developing board, a product of Digilent company. FPGA developing software employed is Xilinx Vivado 2019.1.

Computing results of RTL model are real numbers with long fractional tails. This appearance of results stems from the treatment of real number representation in FPGA, the fixed point representation. Note that results of *PeP* take two significant digits after decimal point. To validate the rightness of RTL model, input NPS1 to *PeP* and compute three steps, Figure 6.6 shows the outcomes. It costs 0.011703 seconds to compute 3 steps of NPS1, outputing $rw = 289.3$ and $lw = 310.7$. So the results of NPS1 RTL model are correct when only two significant digits are preserved. Data accuracy can be improved by assigning more bits to fractional part of a real number variable.

Mark variables to be debugged in Verilog codes and synthesize RTL model of NPS1, then set physical and timing constraints. The I/O planning is illustrated in Figure B.4 in which gray circles (normal input-output pins) and light blue hexagons (clock capable pins) with orange bars inside are pins distributed to RTL model ports. For vector ports (more than one bit), each bit should be assigned a pin. Clock variable is a 1-bit scalar which demands one pin.

Post implementation timing simulation waveform of NPS1 is shown in Figure 6.7. The steady value of the right wheel which is the result of the third step of NPS1 appears at 134.271 ns. Clock period instead of running time is used to calculate speedup because circuits hold values to wait for another clock trigger edge. The

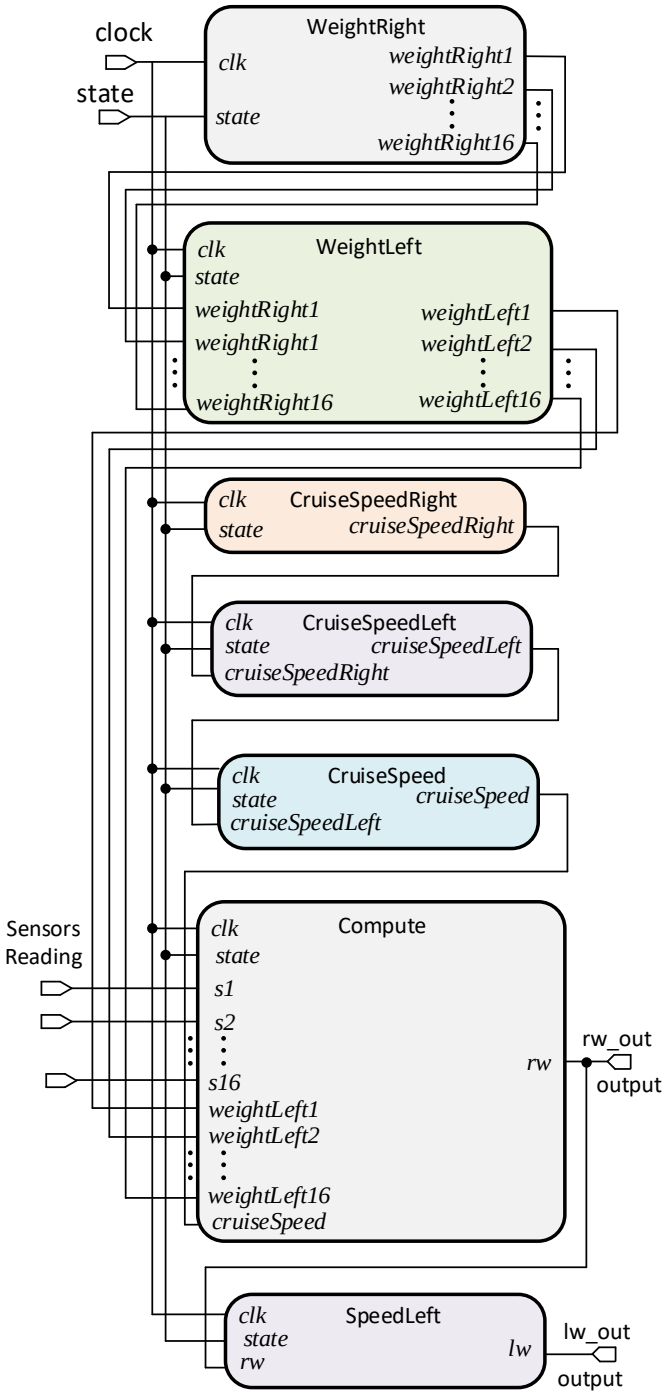


FIGURE 6.4: RTL model of NPS1 consists of 7 modules, although NPS1 has 69 membranes. There is a one-to-one correspondence between membranes and programs inside. This correspondence transforms the implementation of a membrane to implementation a set of programs inside. Programs can be synchronized in one module with parallel constructs of Verilog. This is the reason why the number of modules can be reduced substantially. A *state* port is added to NPS1 so that it possesses idle and busy state.

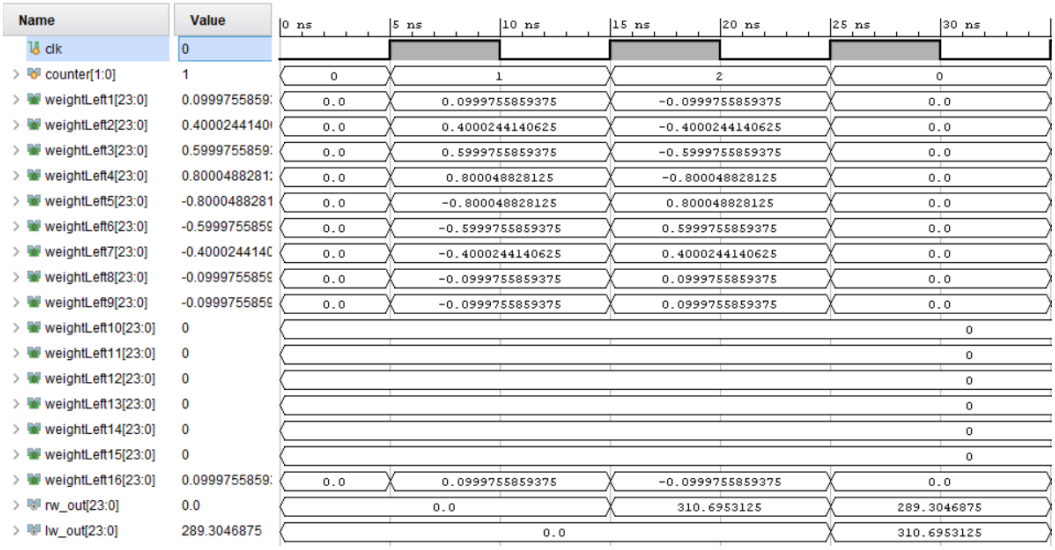


FIGURE 6.5: Waveform of behavioral simulation of NPS1. The values of $weightLeft_i$ alternate as expected when *counter* loops its value. Sensors' readings s_i take the value in the last row of Table 6.1 which are abridged from waveform for the sake of taking a screenshot including computing results *rw-out* and *lw-out*.

```
WARNING:Maximum number of simulation steps exceeded; Simulation stopped
INFO:Simulation finished succesfully after 3 steps and 0.009995 seconds; End state below:
num_ps = {
  SpeedLeft:
    var = { lw: 310.70, }
    E = {}
  SpeedRight:
    var = { rw: 289.30, }
```

FIGURE 6.6: Results of *PeP* simulation of NPS1. It costs 0.011703 seconds to obtain results.

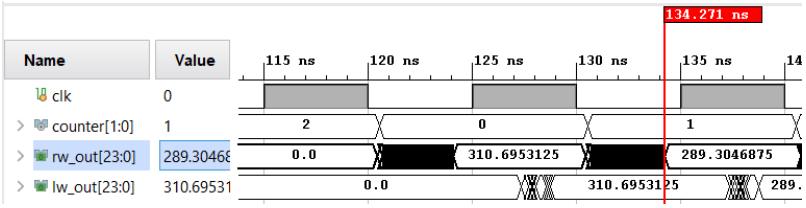


FIGURE 6.7: Post implementation timing simulation of NPS1.

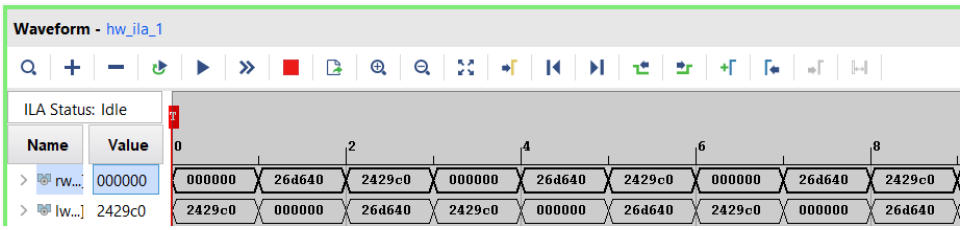


FIGURE 6.8: Hardware debug of NPS1. Numbers are represented in hexadecimal.

speedup of FPGA hardened NPS1 comparing to software simulation is calculated in Formula 6.2. It's pointed out that the computing speed of FPGA hardened NPS1 is 10^8 step per second because clock period is 10 ns with a frequency of 100 MHz. Hardware debug results of NPS1 is given in Figure 6.8. Hardware resource and power consumption of hardened NPS1 is shown in Figure B.5.

$$\frac{9.995 \times 10^6}{1.34271 \times 10^2} \approx 7.4439 \times 10^4 \tag{6.2}$$

6.2 FPGA implementation of membrane controller based on ENPS

ENPS allows multiple programs contained in one membrane to execute concurrently, imparting a feature that can simplify membrane structures. ENPS1 illustrated in Figure 6.9 has the same function as NPS1, but composed of 17 membranes (69 membrans in NPS1). More importantly, ENPS1 computes only one step to get result, improving performance by three times comparing to NPS1 which calculates three steps. Performance improvement is achieved by getting rid of delivery membranes and the speed of left wheel and right wheel are calculated at the same time, not in sequential as what NPS1 does.

Each programs involves a conditional statement and its consequential judgment determines the enforceability of every program in ENPS1. Whereas the values of associated enzymes are tuned so that all the programs can carry out simultaneously. These conditional statement can be transformed to logic expressions. The behavioral simulation ENPS1 RTL model is depicted in 6.10. *PeP* simulation results of ENPS1 is given in Figure 6.11.

Mark variables to be debugged in Verilog codes and synthesize RTL model of ENPS1, then set physical and timing constraints. The I/O planning is illustrated in Figure B.6. Post implementation timing simulation is given in Figure 6.12. Because clock cycle is defined by input half period in the testbench, if the period of clock is an odd number, the half period has a fractional part which will be rounded by Vivado automatically. The clock period of FPGA hardened ENPS1 is 5 ns so its half

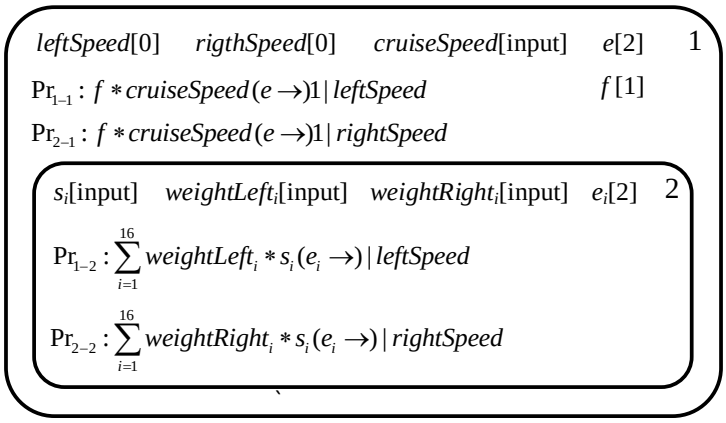


FIGURE 6.9: Enzymatic numerical P system ENPS1. The value of enzyme e is larger than that of f in membrane 1, so these two programs can take place. Enzymes e_i have greater values than weight values so the 16 programs in membrane 2 can execute in parallel.

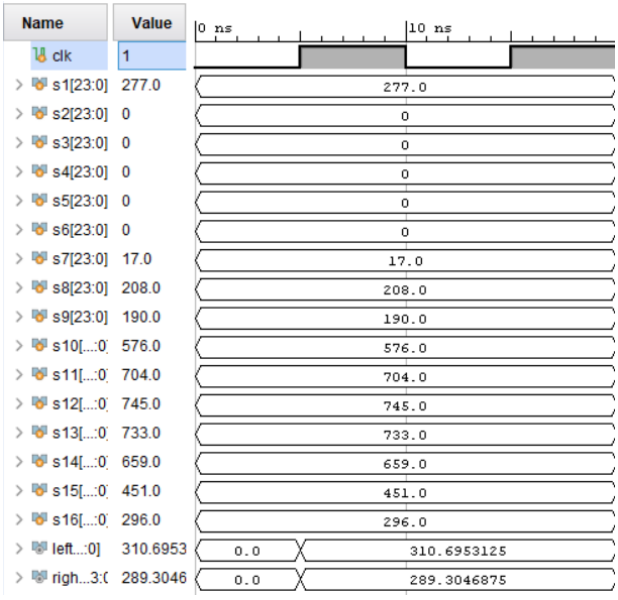
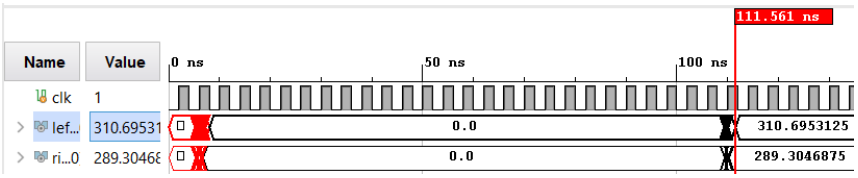


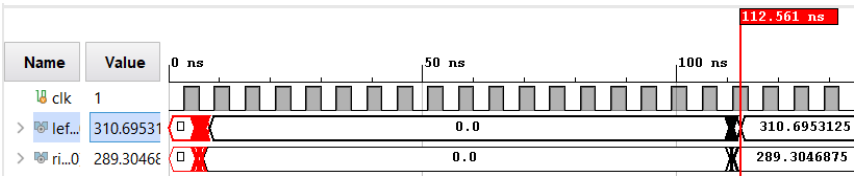
FIGURE 6.10: Waveform of behavioral simulation of ENPS1. Left and right wheel speed variables gain their expected values after the first rising edge.

```
WARNING:Maximum number of simulation steps exceeded; Simulation stopped
INFO:Simulation finished succesfully after 1 steps and 0.002993 seconds; End state below:
num_ps = {
  Avoid:
    var = { Spl: 310.70,  Spr: 289.30,  cruiseSpeedLeft: 0.00,  cruiseSpeedRight: 0.00,  f: 0.00, }
    E = { e: 2.00, }
```

FIGURE 6.11: *PeP* simulation results of ENPS1. It costs 0.002993 seconds to obtain results.



(a) Post implementation timing simulation of ENPS1 with 4 ns period.



(b) Post implementation timing simulation of ENPS1 with 6 ns period.

FIGURE 6.12: Post implementation timing simulation waveform of ENPS1. 112 ns is regarded as the elapsed time to get results.

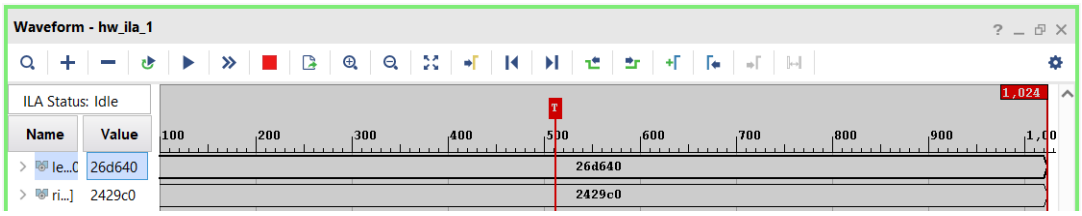


FIGURE 6.13: Hardware debug of ENPS1.

(2.5 ns) cannot be defined. As what have done in the post implementation timing simulation of reaction systems, clocks with period of 4 ns and 6 ns are used in the post implementation timing simulation to reflect the performance of ENPS1 (112 ns).

The speedup of FPGA hardened ENPS1 comparing with *PeP* software simulation is calculated in Formula 6.3. Hardware debug results of ENPS1 is given in Figure 6.13. The hardware resource and power consumption of hardened ENPS1 is shown in Figure B.7. The computation speed of FPGA is 2×10^8 step per second for the clock period is 5 ns. The speedup decreases for the computation task is easy so CPU can also completes in a shorter time, although the clock frequency is 1 time higher than that of FPGA hardened NPS1.

$$\frac{2.993 \times 10^6}{1.12 \times 10^2} \approx 2.67232 \times 10^4 \tag{6.3}$$

6.3 FPGA implementation of membrane controller based on GNPS

The control law 1 presented in Formula 6.1 is targeted at e-puck robot with small a dimension (diameter: 70 mm) and low velocity (max speed: 0.129 m/s) [238]. When transplants this control law to Pioneer 3 DX with dimension of 510×380 mm and cruise speed of 0.4 m/s [268], it is not fit quite well in the new platform. This is because that control law 1 neglects the dimension of robot and angular velocity is a constant which gives rise to a high chance to bump on obstacles when the robot is close to obstacles.

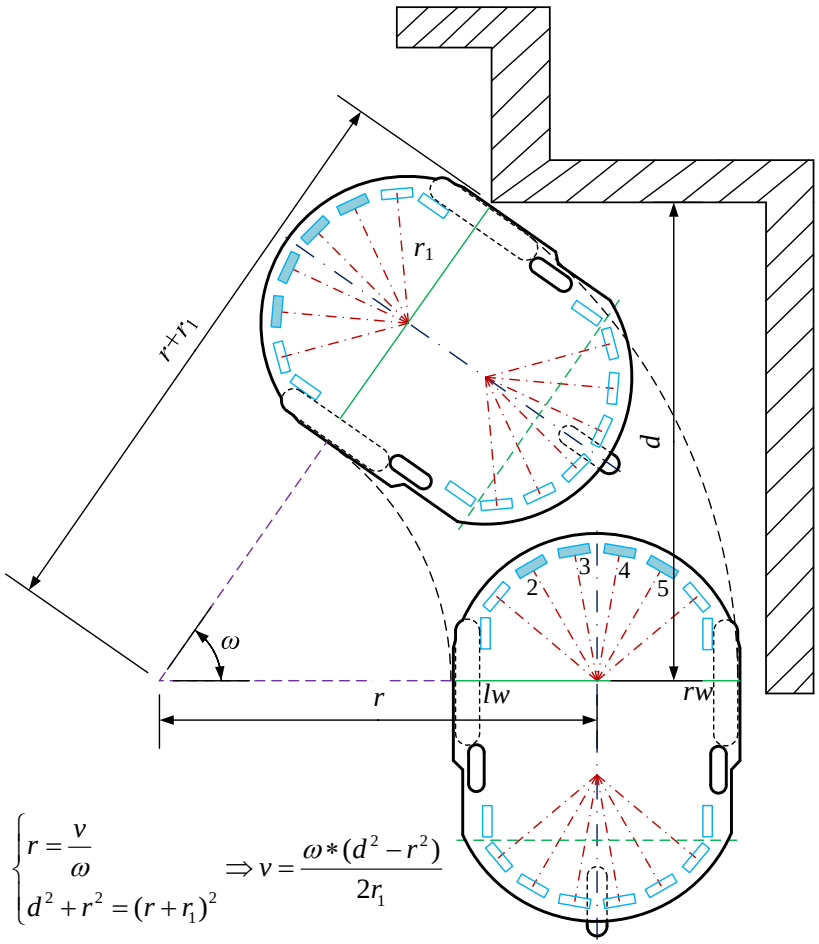


FIGURE 6.14: Obstacle avoidance kinetics analysis of Pioneer 3 DX robot.

Based on kinetics analysis portrayed in Figure 6.14, control law 1 was extended to control law 2 presented in Formula 6.4 [269], where x_i are readings of sonar sensor $i = 2,3,4,5$, r_1 is the robot gyration radius, d_i are the concerned distances detected by sensors, min selects the nearest distance, $H \in (0,1)$ is a safety factor, ω is the angular velocity of robot, θ_i are the angles between symmetric line of robot and the connecting line of robot–obstacles. Other variables have the same meaning as that of control law 1. It is pointed out that control law 1 and 2 can only used in the environment with obstacles higher than the ground. If there is a gap in the ground, these two laws cannot guide the robot the shun it. GNPS3 is contrived to accommodate control law 2, as illustrated in Figure 6.15.

$$\begin{cases} d_i = (x_i + r_1)\cos\theta_i (i = 2,3,4,5) \\ d = min(d_2, d_3, d_4, d_5) \\ CruiseSpeed = H * \frac{|\omega|(d^2 - r_1^2)}{2r_1} \\ lw = CruiseSpeed + \sum_{i=1}^{16} s_i * weithtLeft_i \\ rw = CruiseSpeed + \sum_{i=1}^{16} s_i * weithtRight_i \\ s_i = -x_i + M \end{cases} \tag{6.4}$$

Employ the same hardware facilities as before to implement GNPS3 in FPGA using the method proposed in Chapter 5. The I/O planning is shown in Figure B.8.

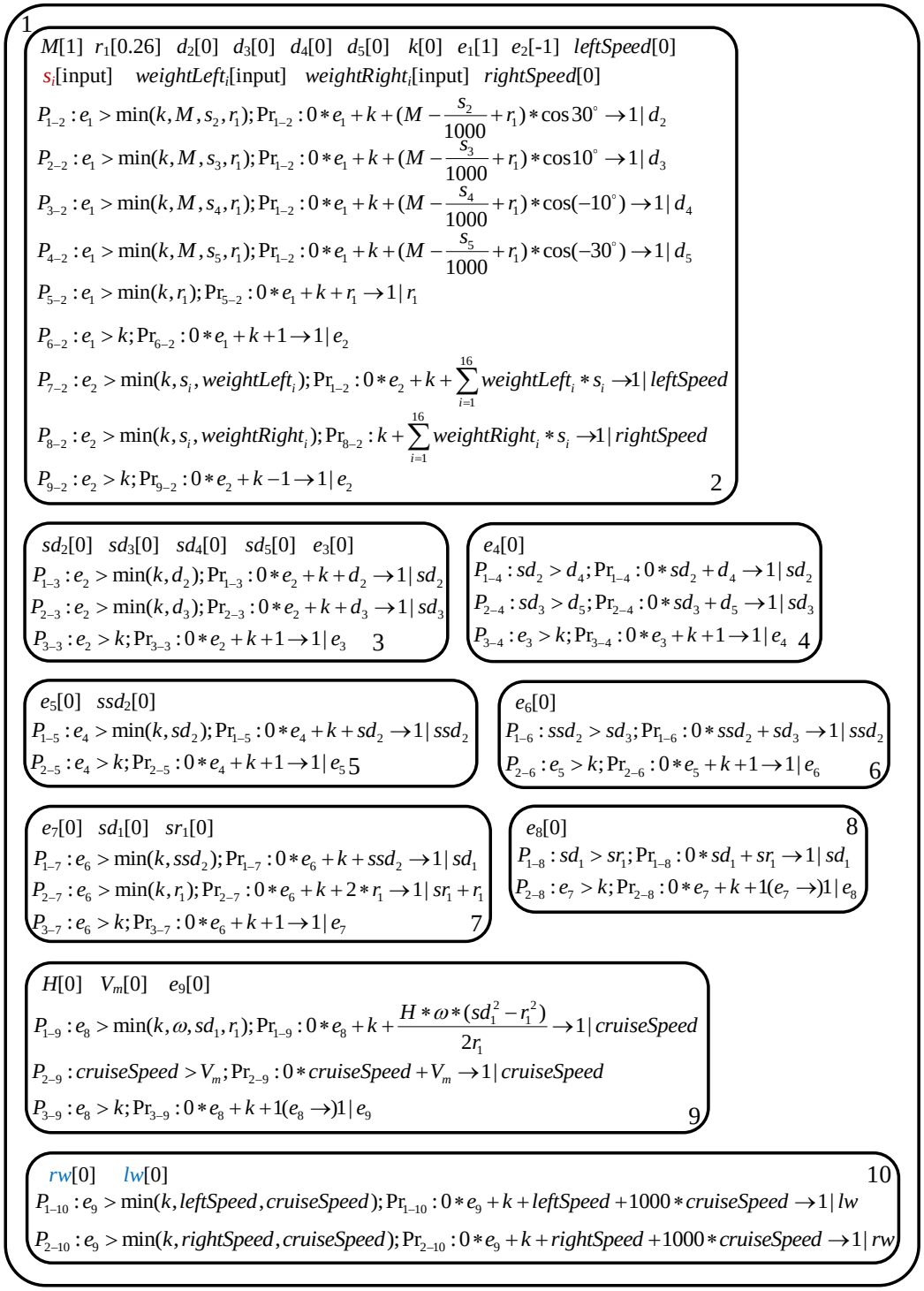
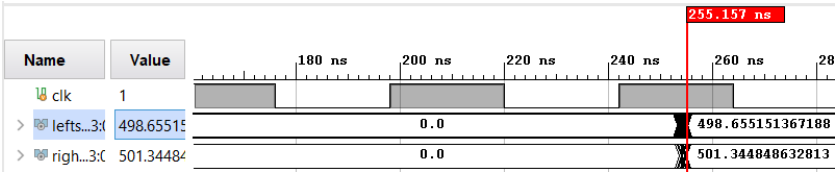
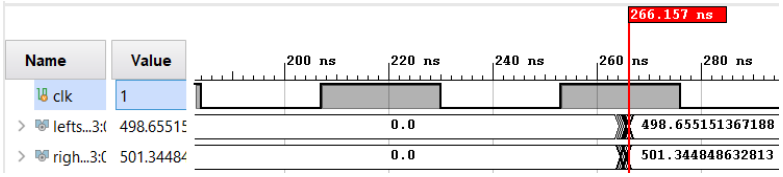


FIGURE 6.15: GNPS3 which carrys out control law 2.



(a) Post implementation timing simulation of GNPS3 with period of 44 ns.



(b) Post implementation timing simulation of GNPS3 with period of 46 ns.

FIGURE 6.16: Post implementation timing simulation of GNPS3. The implemented clock period is 45 ns whose half period cannot defined so 44 ns and 46 ns are set as periods. So the elapsed time is the mean time which is 260.657 ns.

```
WARNING:Maximum number of simulation steps exceeded; Simulation stopped
INFO:Simulation finished succesfully after 10 steps and 0.020997 seconds; End state below:
num_ps = {
  Calculate_Speed:
    var = { Spl: 498.65,  Spr: 501.35,  SWL: 0.00,  SWR: 0.00, }
    E = { e10: 0.00,  cruiseSpeed: 0.00, }
  CruiseSpeed:
    var = { w: 0.00,  Vmax: 0.50,  H: 0.00, }
    E = { e8: 0.00,  e9: -1.00, }
  Calculate_dmin:
    var = {}
    E = {}
  Calculate1:
    var = { T: 0.00,  r1: 0.00,  d2: 865.50,  d3: 984.08,  d4: 984.08,  d5: 865.38,  f: 0.00, }
    E = { e1: 0.00, }
```

FIGURE 6.17: *PeP* simulation of GNPS3. It costs 0.020997 s to output results.

Post implementation timing simulation and *PeP* software simulation are given in Figure 6.16 and 6.17 respectively. The speedup is calculated in Formula 6.5. Hardware debug of GNPS3 is shown in Figure 6.18. Hardware resource and power consumption of FPGA hardened GNPS3 is shown in Figure B.9.

$$\frac{2.0997 \times 10^7}{2.60657 \times 10^2} \approx 8.05541 \times 10^4 \tag{6.5}$$

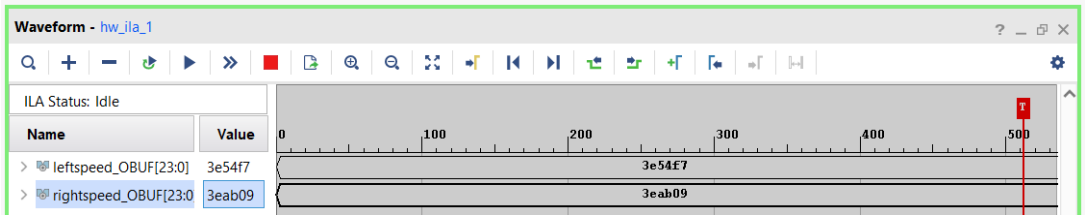


FIGURE 6.18: Hardware debug of GNPS3. Numbers are represented in hexadecimal.

6.4 UART communication of NPS

Substituting the on-board computer of Pioneer 3 DX with FPGA hardened membrane controller to control this robot is the ultimate objective. Pioneer 3 DX is controlled directly by dedicated robot motion controller which is a micro-controller [268] while computing operations are executed by on-board computer. The micro-controller samples reading of sensors and sends them to on-board computer in line with RS232 communication electrical standard protocol. After computation finished, results are transmitted back to micro-controller to control wheel motors. Thereupon, the membrane controller should have a receiver and a transmitter device to receive sensors reading and to send computation results. Universal asynchronous receiver/-transmitter (UART) is such a device meeting RS232 protocol which is employed to design communication devices of membrane controller.

NPS1 is employed as the test bed for the UART communication. The data frame to be communicated has the same format used in Chapter 4: 1 start bit, 8 data bits and 1 stop bits. Data transferring speed of Pioneer 3 DX is set as 115200 baud (nearly 115200 Hz), which should be the working rate of UART receiver and transmitter. While the clock frequency of BASYS 3 FPGA developing board is set as 100 MHz, a module named **frequency splitting** is designed to generate required clock frequency according to the method used in Chapter 4, so as the design of UART receiver and transmitter.

UART receiver can only receive 8-bit data while a sensor reading is assigned a 24-bit data. Therefore it is a triple receiving operation to receive a sensor reading. A register is assigned to store received 8-bit data so that 48 registers are needed to store sixteen 24-bit data of sensors' reading. Values in three adjacent register are concatenated to form a 24-bit binary value which is a input data to NPS1. NPS1 has two operating state: idle and busy state. Before NPS1 receiving sensor readings, it is in idle otherwise it is in busy after fed to full. The storing rate is $\frac{1}{10}$ of receiving frequency for 10 bits are sent sequentially while computing frequency should be $\frac{1}{30}$ of receiving frequency since it performs 3 times of receiving operation to get a entire sensor reading. The working frequency of UART transmitter equals to the storing rate for it sends 10 serial bits to micro-controller which works in 115200 Hz. The RTL model of UART-NPS1 is illustrated in Figure 6.19.

According to the computational process of NPS1, the first output of *rw* is zero and the latter two outputs are expected values. UART transmitter should skip zero and begin to transmit when the first result arise. So transmitter stays in idle before *rw* is nonzero. Wheel speed values to be transmitted are 24-bit data, but each time the transmitter can send only 8 bits. Hence the transmitter sends three times successively to transmit 24 bits in total.

UART communication experiment between host computer and BASYS 3 is conducted to verify proposed design method. In this experiment, host computer sends sixteen 24-bit sensor readings to FPGA via integrated UART transmitting port within it and BASYS 3 receives these data into forty eight 8-bit registers. Every three 8-bit registers are concatenated to construct a 24-bit register so that data inside can be 24-bit. After computation of FPGA NPS1, FPGA transmits results to host computer and these values can be observed on the screen. Realterm is the serial debugging software adopted to configure ports setting and display data from FPGA. It is obvious that FPGA NPS1 computes and transmits correctly according to Figure 6.20 which shows the experiment result.

A notable phenomenon can be found in Figure 6.20 is that UART transmitter does not stop transmitting so data strings repeat. The transmitting counter will enter

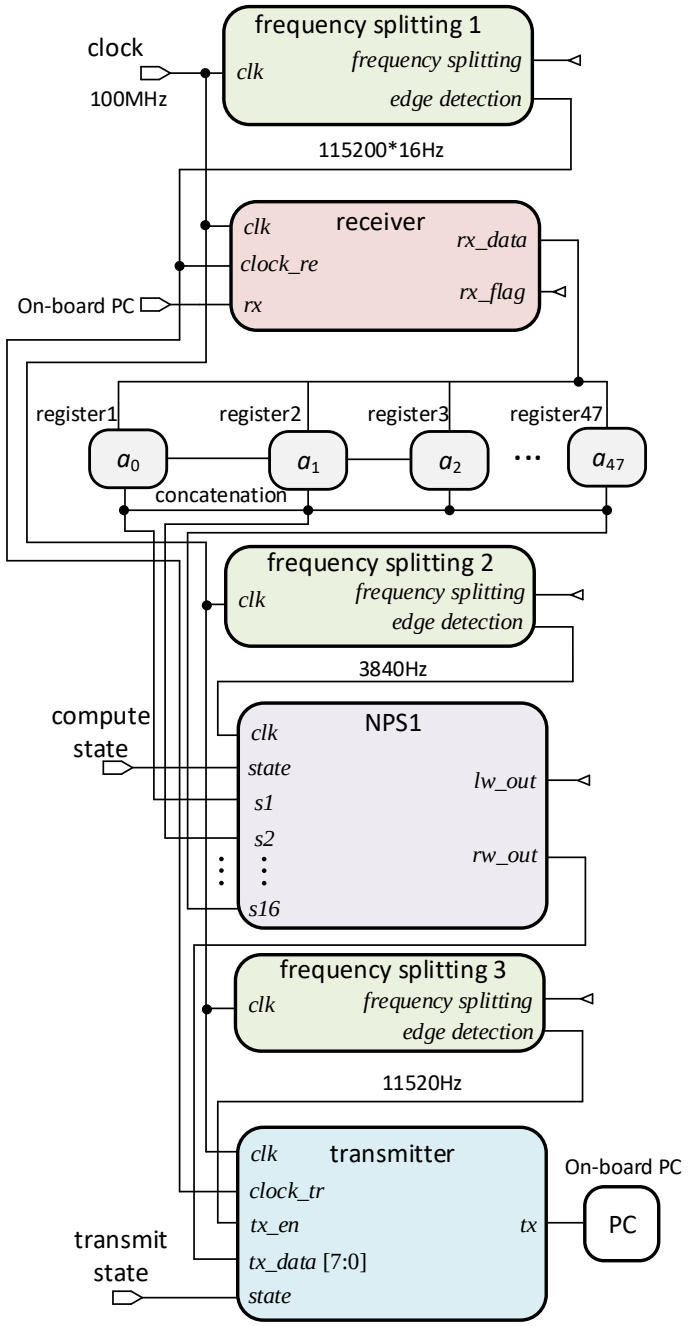


FIGURE 6.19: RTL model of UART-NPS1.

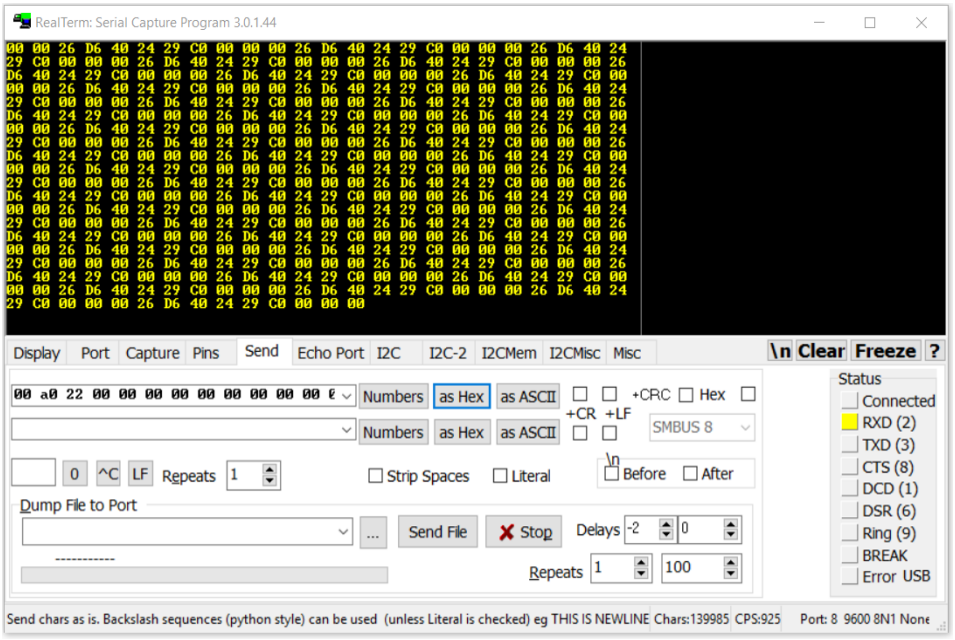


FIGURE 6.20: Numbers are in hexadecimal for real numbers are represented in fixed point binary numbers. These data repeats 00 00 00, 26 D6 40 and 24 29 C0. Two digits of hexadecimal correspond to 8 bits in binary so three dual hexadecimal pairs comprise a 24-bit binary number. These three repeated data strings are 0, 310.6953125 and 289.3046875 in decimal, which are the desired three outcomes of *rw* of NPS1.

infinite loop if clock does not stop. In effect, any counter will loop forever unless it is related to some limited signals. Input signals are limited because they will stop at a time. Transmitting counter should come into play to count from 0 only after all sensor readings are received and stop counting when it counts to 5 (corresponding to the latter two value of *rw* which are six 8-bit binary data). By this way, transmitting counter will not loop and transmitter sends six 8-bit binary data. The effectiveness of this strategy is demonstrated by the waveform of UART-NPS1 RTL model in Figure 6.21.

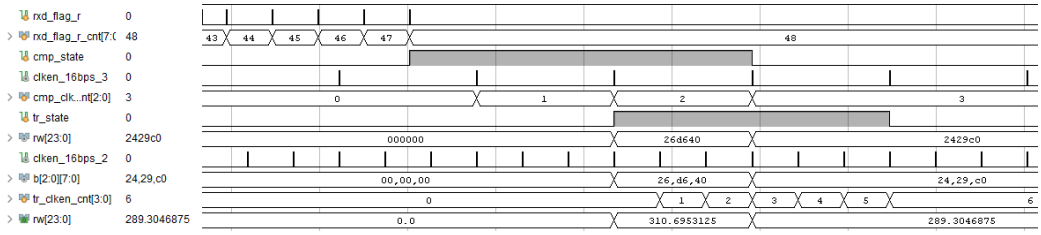


FIGURE 6.21: When input counter counts from 0 to 47, sensor readings are received into NPS1. Computing signal *cmp-state* converts from 0 to 1 to trigger computing. When left wheel speed are outputted, transmitting signal *tr-state* shifts from 0 to 1 to trigger transmitting. At the time when transmitting counter signal *tr-clken-cnt* has value of 5, *tr-state* switches its value to terminate transmitting.

6.5 Wrap-up

Membrane controllers carrying out control law 1 and 2 are implemented in FPGA based on NPS, ENPS and GNPS, achieving a speedup of a order of 10^4 comparing with software simulation. The theoretical parallelism of P systems are obtained and deployed in FPGA which can substitutes the on-board computer of robot with the supplement of UART communication component tailored to target membrane controller. ENPS can simplify membrane structures to a large extend for the special function of enzymatic variables. GNPS can perform complicated operations for its structural/functional advantages.

Chapter 7

FPGA Architecture for Generalized Numerical P System modeled Rapidly-exploring Random Tree Algorithm

This chapter investigates GNPS based FPGA implementation of an intensive computation algorithm in robot path planning–rapidly-exploring random tree (RRT) algorithm. This algorithm commence at a predefined initial point O. A random point R_1 is generated whose two coordinates are products of initial point coordinates and two random numbers in the range of $[0, 1]$. Then compute a point S_1 in the line segment OR_1 so that $|OS_1| = \delta$. Next calculate all the distances between obstacle points and line segment OS_1. If all the distances are larger than the rotation radius of robot, save point S_1 as the first RRT point, otherwise discard it and re-calculate R_1. After a satisfied S_1 obtained, compute the second random point R_2. Find the nearest point N_1 from O and S_1 ($N_1 \in \{O, S_1\}$). Determine point S_2 in the line segment N_1R_2 so that $|N_1S_2| = \delta$. Calculate all the distances between obstacle points and line segment N_1S_2 to identify whether S_2 is the second RRT point. By this way, one can compute more RRT points. The path generated by connecting all RRT points is object free.

The first attempt to arrange RRT in ENPS is presented in [270], where a variant of ENPS named random enzymatic numerical P systems with proteins and shared memory is designed to organize RRT. This model is simulated with a extended P-Lingua software [271]. A hierarchical FPGA architecture for RRT algorithm is given in [272] while a hybrid architecture composed of combinatorial and hierarchical architectures is proposed in [273, 274]. In [275], a FPGA parallel architecture is devised for a variant of RRT, the RRT*, achieving speedups from 30∼90 times comparing embedded/desktop software simulation.

7.1 Rapidly-exploring random tree algorithm

As a randomized planning technique, rapidly-exploring random tree (RRT) has several good qualities such as it is biased to unexplored state space, the vertices (referred to as RRT points in this paper) are nearly uniformly distributed, and only nearest-neighbor queries are needed [276, 277], ect. RRT has had been increasingly applied in path planning since its establishment in 1998. There are many researches on this topic, refers to [278, 279, 280, 281, 282]. The robot initial point (x_1, y_1) is the root RRT point. Generating the first RRT point (x_2, y_2) is a little bit different from others for it

is the only one RRT point for the time being and is the nearest one to the first random point (x_{rand1}, y_{rand1}). The procedures for producing the first RRT point is given in Algorithm 2.

To produce the second RRT point (x_3, y_3), after the calculation of the secong random point (x_{rand2}, y_{rand2}), compute the distances of the two RRT point (x_1, y_1) and (x_2, y_2) to the random point. Then select the nearest point from them to the random point according to their distances. Proceed the remaining procedures of computing a potential RRT point and its verification, as illustrated in Algorithm 3.

As can be seen, to compute the nth ($n \geq 2$) RRT point, there are n distances between RRT points (including the initial point) and the random point to be computed. The size of the subsequent computation and compare logic determining the nearest distance would also be large if n is a large number. And its size keeps increasing if more RRT points are required. This is a incremental process so RRT belongs to the category of incremental sampling-based motion planning algorithms. If m denotes the number of obstacle points, in the verification process of each iteration, m distances of obstacle points to the line segment $(x_n, y_n) - (x_{new}, y_{new})$ ((x_n, y_n) is the nth RRT point, (x_{new}, y_{new}) is the $n+1$th potential RRT point) should be computed and the nearest distance should be chosen, comparing with robot rotation radius ζ to determine discarding or storing the potential point (x_{new}, y_{new}).

The procedures of finding out the nearest RRT point to random point and identification process of potential RRT points can be performed in parallel. The processing speed will be improved to a large extend if RRT algorithm is executed in a parallel hardware architecture. IEEE 754 floating point (FP) number is selected as the real number format for its large dynamic range and high precision. This format allows the future application of RRT into a large scale map with an amount of obstacle points. Taking into account of the complexity of RRT algorithm and IEEE 754 format, and the capacity of the target FPGA chip, a RRT generating 2 RRT points in an environment with 8 obstacle points, diagrammatically shown in Figure 7.1, is modeled in a generalize numerical P system (GNPS) and implemented in a Xilinx FPGA. The value of this research lies in that a FPGA implementation method of GNPS working with floating point number is proposed and a new approach to implement RRT algorithm on FPGA.

Despite the simplicity of the RRT considered, the method proposed is easy to scale to incorporate more obstacle points and generate more RRT points, provided that target FPGA has enough hardware resources. The GNPS dealing with floating point (FP) numbers which generates two RRT points in an environment with eight obstacle points is devised, as illustrated in Figure 7.2. Results of program usability predicates play the role of metronomes to make GNPS working in sequential or in parallel in line with RRT algorithm. GNPS with a signature containing the inverse square root function is used. The floating point (FP) inverse square root unit is designed to calculate programs associated with potential RRT point generation (e.g., program Pr_{6-1}, Pr_{7-1}). So division operations involved in these programs are averted. For those divisions cannot be shunned (e.g., program Pr_{6-2}, Pr_{5-9}, etc.), Xilinx FP divider IP cores are instantiated to perform.

For the sake of confirming the correctness of RRT-GNPS, the model's ENPS counterpart is emulated in *PeP* [243]. Part of the simulation results is given in Figure 7.3 in which the expected results are obtained. *PeP* requires that every membrane contains at least one enzyme. So contents of membrane 3 to 9 which do not have enzymatic variables e_i, $i = 3, \ldots, 9$, are put into membrane 2 during the simulation. *PeP* cannot execute programs like Pr_{13-2}, Pr_{12-9} and Pr_{7-18} as well, so which variables should

Algorithm 2: RRT algorithm generating the first RRT point.

Input: $p, q, \delta, \xi, (x_1, y_1), rand_i, i = 1, 2$
Output: (x_2, y_2)

1 $(x_{rand1}, y_{rand1}) \leftarrow ((p * rand_1), (q * rand_2))$;
$d_1 \leftarrow (x_1 - x_{rand1})^2 + (y_1 - y_{rand1})^2$;
$(x_{new1}, y_{new1}) \leftarrow (x_1 + \frac{\delta * (x_{rand1} - x_1)}{\sqrt{d_1}}, y_1 + \frac{\delta * (y_{rand1} - y_1)}{\sqrt{d_1}})$;
$d_{sr1} \leftarrow (x_{new1} - x_1)^2 + (y_{new1} - y_1)^2$;
2 **for** $j = 1; j < N_o + 1; j = ++$ **do**
3 $u_{1j} \leftarrow (x_{obj} - x_1)(x_{new1} - x_1) + (y_{obj} - y_1)(y_{new1} - y_1)$;
$d_{noj} \leftarrow (x_1 - x_{obj})^2 + (y_1 - y_{obj})^2$; $d_{nbj} \leftarrow (x_{new1} - x_{obj})^2 + (y_{new1} - y_{obj})^2$;
4 **end**
5 **for** $j = 1; j < N_o + 1; j = ++$ **do**
6 $u_{2j} \leftarrow \frac{u_{1j}}{d_{sr1}}$;
7 **end**
8 **for** $j = 1; j < N_o + 1; j = ++$ **do**
9 $p_{xj} \leftarrow u_{2j} * (x_{new1} - x_1)$; $p_{yj} \leftarrow u_{2j} * (y_{new1} - y_1)$;
10 **end**
11 **for** $j = 1; j < N_o + 1; j = ++$ **do**
12 $d_{poj} \leftarrow (p_{xj} - x_{obj})^2 + (p_{yj} - y_{obj})^2$;
13 **end**
14 **for** $j = 1; j < N_o + 1; j = ++$ **do**
15 **if** $u_{2j} < 0$ **then**
16 $psd_j \leftarrow d_{noj}$;
17 **end**
18 **else if** $u_{2j} > 1$ **then**
19 $psd_j \leftarrow d_{nbj}$;
20 **end**
21 **else**
22 $psd_j \leftarrow d_{poj}$;
23 **end**
24 **end**
25 $d_{ls} \leftarrow min(psd_1, \ldots, psd_8)$;
26 $collision \leftarrow (\xi - d_{ls})$;
27 **if** $collision < 0$ **then**
28 $(x_2, y_2) \leftarrow (x_{new1}, y_{new1})$;
29 **end**
30 **else**
31 back to line 1;
32 **end**

Algorithm 3: RRT algorithm generating the second RRT point.

Input: $p, q, \delta, \xi, (x_1, y_1), (x_2, y_2), rand_i, i = 1, 2$
Output: (x_3, y_3)

1 $(x_{rand2}, y_{rand2}) \leftarrow ((p * rand_1), (q * rand_2))$;
$d_{rt1} \leftarrow (x_1 - x_{rand2})^2 + (y_1 - y_{rand2})^2$; $d_{rt2} \leftarrow (x_2 - x_{rand2})^2 + (y_2 - y_{rand2})^2$;
2 **if** $d_{rt1} < d_{rt2}$ **then**
3 $(x_{nearest2}, y_{nearest2}) \leftarrow (x_1, y_1)$;
4 **end**
5 **else**
6 $(x_{nearest2}, y_{nearest2}) \leftarrow (x_2, y_2)$;
7 **end**
8 $d_1 \leftarrow (x_{nearest2} - x_{rand2})^2 + (y_{nearest2} - y_{rand2})^2$;
$(x_{new2}, y_{new2}) \leftarrow (x_{nearest2} + \frac{\delta * (x_{rand2} - x_{nearest2})}{\sqrt{d_1}}, y_{nearest2} + \frac{\delta * (y_{rand2} - y_{nearest2})}{\sqrt{d_1}}$;
$d_{sr2} \leftarrow (x_{new2} - x_{nearest2})^2 + (y_{new2} - y_{nearest2})^2$;
9 **for** $j = 1; j < N_o + 1; j = ++$ **do**
10 $u_{1j} \leftarrow$
$(x_{obj} - x_{nearest2})(x_{new1} - x_{nearest2}) + (y_{obj} - y_{nearest2})(y_{new1} - y_{nearest2})$;
$d_{noj} \leftarrow (x_{nearest2} - x_{obj})^2 + (y_{nearest2} - y_{obj})^2$;
$d_{nbj} \leftarrow (x_{new2} - x_{obj})^2 + (y_{new2} - y_{obj})^2$;
11 **end**
12 **for** $j = 1; j < N_o + 1; j = ++$ **do**
13 $u_{2j} \leftarrow \frac{u_{1j}}{d_{sr2}}$;
14 **end**
15 **for** $j = 1; j < N_o + 1; j = ++$ **do**
16 $p_{xj} \leftarrow u_{2j} * (x_{new2} - x_{nearest2})$;
17 $p_{yj} \leftarrow u_{2j} * (y_{new2} - y_{nearest2})$;
18 **end**
19 **for** $j = 1; j < N_o + 1; j = ++$ **do**
20 $d_{poj} \leftarrow (p_{xj} - x_{obj})^2 + (p_{yj} - y_{obj})^2$;
21 **end**
22 **for** $j = 1; j < N_o + 1; j = ++$ **do**
23 **if** $u_{2j} < 0$ **then**
24 $psd_j \leftarrow d_{noj}$;
25 **end**
26 **else if** $u_{2j} > 1$ **then**
27 $psd_j \leftarrow d_{nbj}$;
28 **end**
29 **else**
30 $psd_j \leftarrow d_{poj}$;
31 **end**
32 **end**
33 $d_{ls} \leftarrow min(psd_1, \ldots, psd_8)$;
34 $collision \leftarrow (\xi - d_{ls})$;
35 **if** $collision < 0$ **then**
36 $(x_3, y_3) \leftarrow (x_{new2}, y_{new2})$;
37 **end**
38 **else**
39 back to line 1;
40 **end**

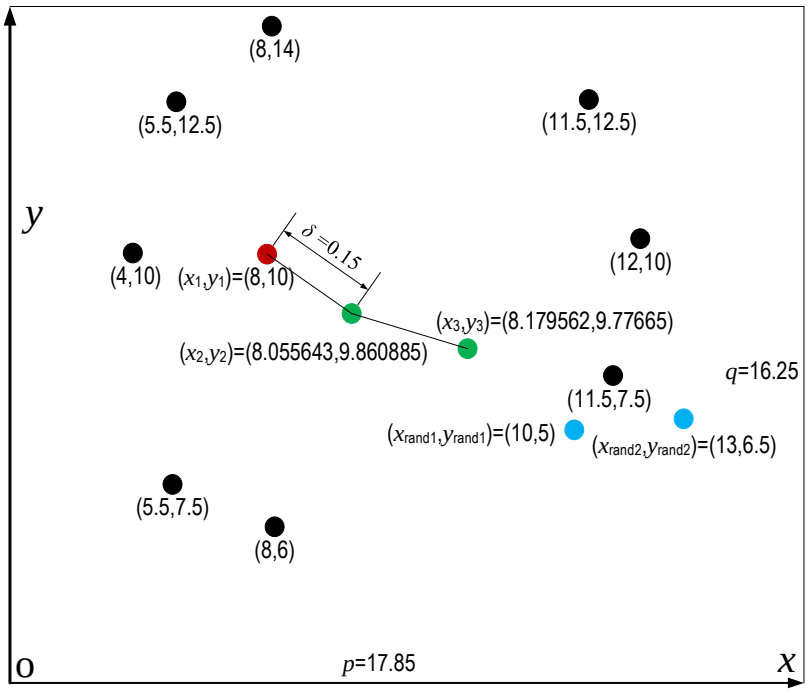


FIGURE 7.1: Graphical representation of the target RRT. Black points indicates obstacle points and the red point denotes robot initial point. Blue points are random points and green points are generated RRT points.

be chosen is determined manually in these programs. RRT-GNPS excludes the random number generator for it is impractical to design such a membrane producing random numbers. Random numbers used in RRT-GNPS are given as constant in the software simulation. For FPGA implementation, an IEEE 754 compliant FP random number generator is designed to feed random numbers.

7.2 Floating point arithmetic units design

7.2.1 Basics of IEEE 754 single precision floating point standard

This research devises IEEE 754 single precision floating point compliant arithmetic units to perform RRT algorithm modeled in GNPS. As can be seen from Figure 7.4(a) which illustrates the format of IEEE 754 single precision floating point standard, 32 bits ($[31:0]$) are assigned to a FP number. Bit 0 to 22 store mantissa and bit 23 to 30 represent exponent. The last bit denotes the sign, 0 for positive and 1 for negative. The exponent is an 8-bit unsigned number, so it cannot represents negative exponent. To resolve this problem, exponent is biased by the constant 127. But it does not mean the exponent range is $[-127, 128]$ since several special cases called *exceptions* utilize the range endpoint. Table 7.1lists these exceptions and normalization.

Mantissa is normalized that there is an 1 at the left hand side of the binary point (for instance 1.0001101101). However, in order to store one more bit to increase precision, this leading 1 is not stored but implied. As a result, all the 23 bits are fractional parts, form 2^{-1} to 2^{-23}. The value of a mantissa is $1 + \frac{mantissa}{2^{23}}$ where *mantissa* is the integer number denoted by the 23-bit of mantissa. This formula is deduced as follows.

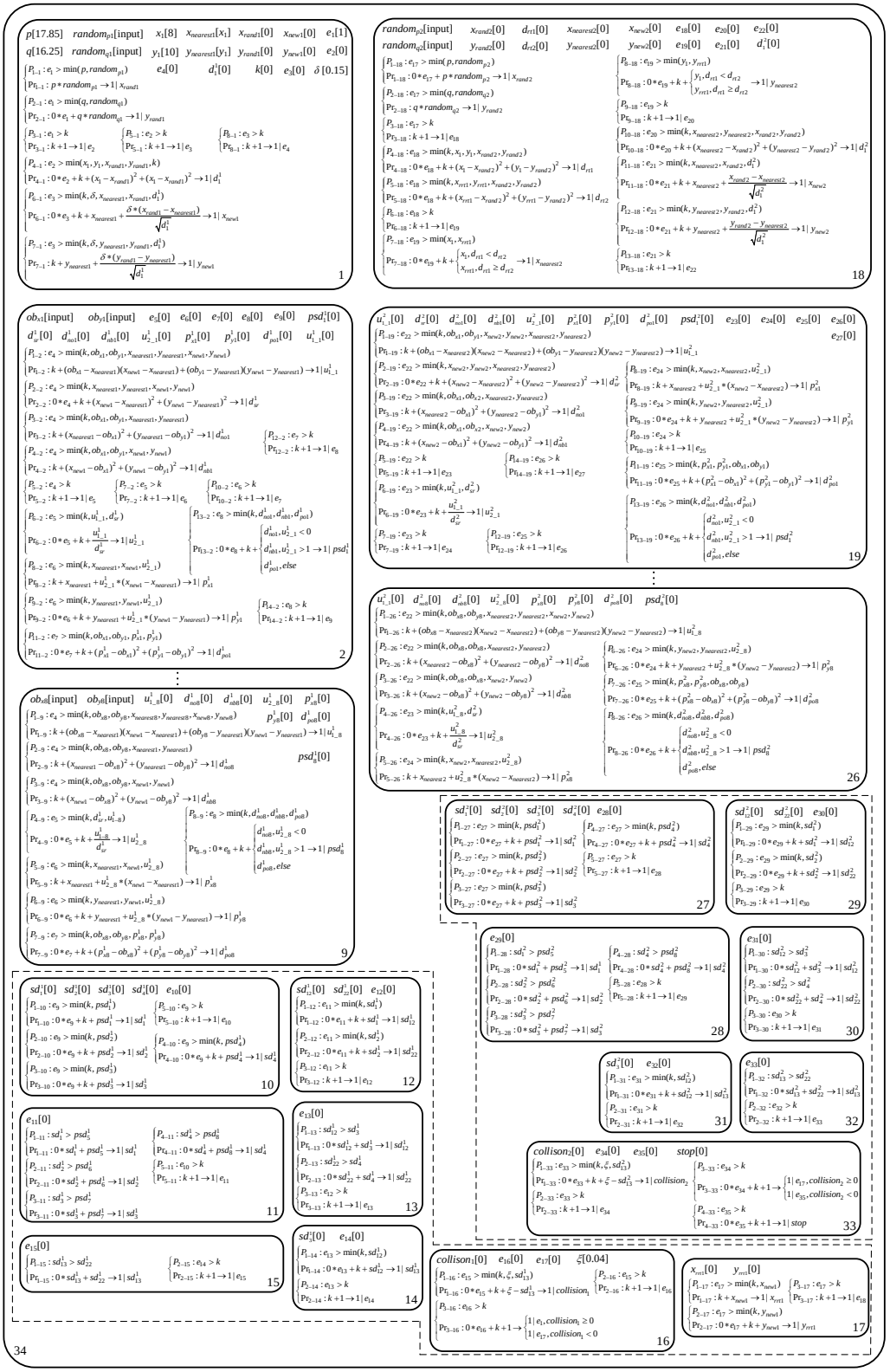


FIGURE 7.2: The RRT-GNPS designed to execute RRT algorithm which generates two RRT points in eight obstacle points. It consists of 34 membranes including the skin membrane.

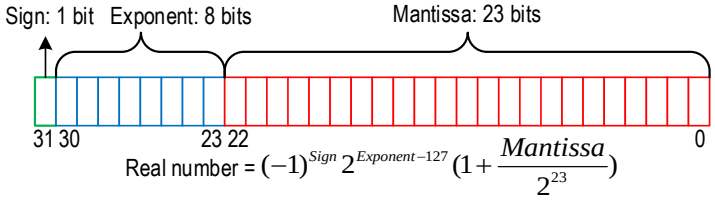
```
WARNING:Maximum number of simulation steps exceeded; Simulation stopped
INFO:Simulation finished succesfully after 34 steps and 0.097948 seconds; End state below:
num_ps = {
  skin:
    var = {}
    E = {}
  mem1:
    var = { p: 0.00,  q: 0.00,  random_p1: 0.00,  random_q1: 0.00,  x1: 8.00,  y1: 10.00,  x_nearest1: 8.00,  y_nearest1
: 10.00,  x_rand1: 10.00,  y_rand1: 5.00,  x_new1: 8.06,  y_new1: 9.86,  k: 0.00,  delta: 0.15,  one_d1: 0.00,  two_p: 0
.00,  two_q: 0.00,  two_x_nearest1: 0.00,  two_y_nearest1: 0.00,  two_delta: 0.00,  two_x_rand1: 0.00,  two_y_rand1: 0.0
0,  two_x1: 0.00,  two_y1: 0.00, }
    E = { e1: 0.00,  e2: 0.00,  e3: 0.00,  e4: 0.00, }
```

(a) RRT-GNPS membrane 1 simulation results which contains the first RRT point $(8.06, 9.86)$.

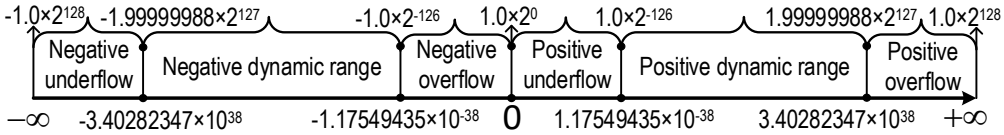
```
  mem18:
    var = { random_p2: 0.00,  random_q2: 0.00,  x_rand2: 13.00,  y_rand2: 6.50,  d_rt1: 37.25,  d_rt2: 35.74,  x_nearest
2: 8.06,  y_nearest2: 9.86,  x_new2: 8.18,  y_new2: 9.78,  two_x_rrt1: 0.00,  two_y_rrt1: 0.00,  two_d1: 0.00,  two_x_ne
arest1: 0.00,  two_y_nearest1: 0.00,  two_x_rand2: 0.00,  two_y_rand2: 0.00,  two_x_nearest2: 0.00,  two_y_nearest2: 0.0
0,  three_x_nearest2: 0.00,  three_y_nearest2: 0.00,  three_x_rand2: 0.00,  three_y_rand2: 0.00, }
    E = { e18: 0.00,  e19: 0.00,  e20: 0.00,  e21: 0.00,  e22: 0.00, }
```

(b) RRT-GNPS membrane 18 simulation results which contains the second RRT point $(8.18, 9.78)$.

FIGURE 7.3: *PeP* simulation of RRT-GNPS which perform 34 steps and costs 0.097948 s to get results. The results of *PeP* save 2 significant digits.



(a) IEEE 754 single precision floating point representation.



(b) Dynamic range of IEEE 754 single precision floating point format.

FIGURE 7.4: The dynamic range is represented using radix 2 as well as radix 10. Zero is denoted by 1.0×2^0, $-\infty$ by -1.0×2^{128} and $+\infty$ by 1.0×2^{128}.

TABLE 7.1: Exceptions and normalization of IEEE 754 single precision floating point number.

Exponent	Mantissa	Number
255	0	$\pm\infty$
0	0	0
1∼254	arbitrary	$\pm normal number$
0	non 0	$\pm denormal number$
255	non 0	not a number (NaN)

$$\begin{aligned} n_1 \times 2^{-1} + \cdots + n_{23} \times 2^{-23} &= \frac{2^{23} \times (n_1 \times 2^{-1} + n_2 \times 2^{-2} + \cdots + n_{23} \times 2^{-23})}{2^{23}} \\ &= \frac{n_1 \times 2^{22} + n_2 \times 2^{21} + \cdots + n_{23} \times 2^{0}}{2^{23}} \\ &= \frac{mantissa}{2^{23}} \end{aligned}$$

where n_i =0 or 1, $i = 1, 2, \ldots, 23$.

Figure 7.4(b) shows the dynamic range based on radix 2 and radix 10. Take the conversion from $1.99999988 \times 2^{127}$ to $3.40282347 \times 10^{38}$ as the example, the dynamic range with different radix can be deduced as below.

$$\begin{aligned} 10^x &= 2^{127} \\ \log_2 10^x &= \log_2 2^{127} \\ x \log_2 2 \times 5 &= 127 \\ x(1 + \log_2 5) &= 127 \\ x &= \frac{127}{1 + \log_2 5} \approx 38.2308094492 \\ 1.99999988 \times 2^{127} &\approx 1.99999988 \times 10^{38.2308094492} \\ 1.99999988 \times 10^{0.2308094492} &\approx 3.40282347 \\ 1.99999988 \times 2^{127} &\approx 3.40282347 \times 10^{38} \end{aligned}$$

7.2.2 Sequential triggered IEEE 754 compliant adder

FP arithmetical expressions cannot be written in Verilog *always* blocks if they are described by a series of instantiated adders, multipliers and other arithmetical units, other than those fixed point format formulas represented explicitly by Verilog operators. If all the arithmetical units are triggered only by clock rising edge, they will always be active and the timing is tremendously chaotic. So these floating point arithmetical units are designed in such way that they are sequentially triggered instead of synchronous triggered. Arithmetic operations are essentially serial operations, dealing first with operations with higher priority (multiplication and division) and then with operations with lower priority (addition and subtraction). For operations with the same priority, they are executed sequentially. Multiple such serial operations can be performed simultaneously in FPGA. A counter is added to count the clock cycle and a flag signal generated by rising edge detection bears a rising edge at the clock rising edge. This flag signal holds value 1 only for one clock period so at the next rising edge of clock, the flag has a falling edge and keeps low. This flag can be used to fire the unit whose trigger signal port connects to this flag just once.

Addition and subtraction are closely related so that $a + b$ does not always mean an addition, it can be a subtraction as well. The ultimate action is determined not only by the operator but also by the signs and magnitudes of both operands. As a consequence, an adder can be a subtractor at the same time. The FP adder is designed in line with this thought. So the adder must have a port to input desired operator, + (binary 1) or - (binary 0). The final operation is the XOR of the sign of two operands and the input operator. The sign of result is deduced in Table 7.2.

TABLE 7.2: Determine the sign of the add/subtract result.

sign of a	operator	sign of b	result sign
+	+	+	+
+	+	-	sign of the operand with larger abs
+	-	+	sign of the operand with larger abs
+	-	-	+
-	+	+	sign of the operand with larger abs
-	+	-	-
-	-	+	-
-	-	-	sign of the operand with larger abs

FP addition/subtraction requires that exponents of the two operands should be equal. If it is not the case, we should fix one as baseline, the bigger one generally, scale the exponent of smaller one and shift its mantissa accordingly. The main procedures of the adder is diagrammatically shown in Figure 7.5. To reduce hardware resources and power consumption, exceptions are not taken into account except underflow. Truncation is selected as the rounding mode [283, 284] for the same reason. According to the difference of trigger time, two types of adder are designed: *Fp_add_Egs* which computes at the clock rising edge when its trigger flag signal is in high level; *Fp_add_Egsc* works at the same condition as *Fp_add_Egs* but lags behind this adder two clock period.

7.2.3 Sequential triggered IEEE 754 compliant multiplier and FP comparator

The design of floating point multiplier is relatively easy comparing to the design of a floating point adder. Add the exponents of two operands and multiply their mantissas, then extract specific 23-bit according to IEEE 754 format. The main procedures of the multiplier is illustrated in Figure 7.6. There are four types of multipliers are devised according to trigger time. *Fp_mul_Eg* has no additional trigger port and works at rising edge of clock. The trigger time of *Fp_mul_Ess* is the same as that of *Fp_add_Egs*. *Fp_mul_Es* computes one clock cycle later than *Fp_mul_Ess*. While *Fp_mul_Esc3* delays three cycles comparing to *Fp_mul_Ess*. A reset port is designed in adders and multipliers to restore them to initial state to begin another calculation.

It is trivial to compare magnitude of two fixed point numbers for we can use Verilog $\geq$ and $\leq$ operator. However to compare two floating point numbers is not such intuitive because we have to design comparator, no operators to use. There are three parts in a floating point number: sign, exponent and mantissa. At first the sign bit is compared then comparing exponent and mantissa successively. Two FP numbers are equal only when these three parts are equal correspondingly. Absolute value comparison which combines exponent and mantissa compare can simplify this process to some extend. This is the trick adopted to design the comparator that compares two input FP numbers and output the smaller one, see Table 7.3.

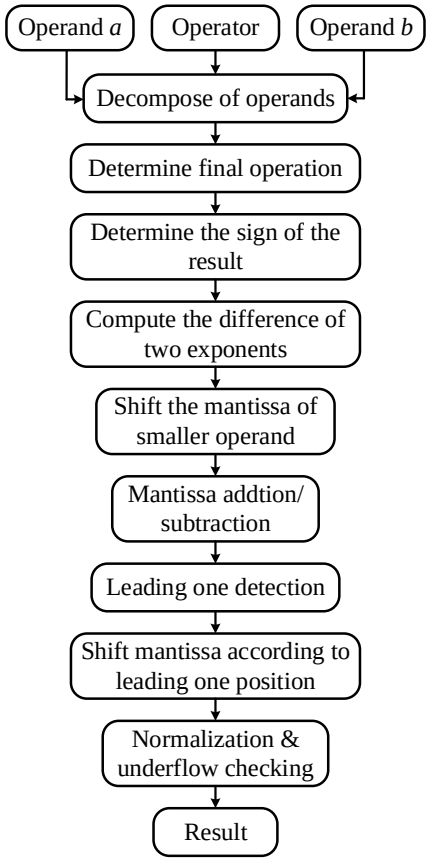


FIGURE 7.5: Procedure diagram of the adder. This is a composite unit which can perform addition and subtraction.

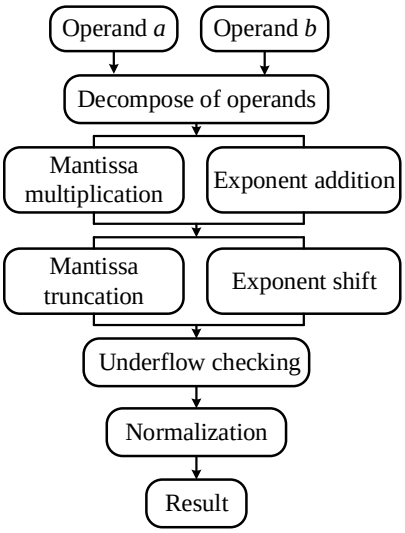


FIGURE 7.6: Procedure diagram of the multiplier.

TABLE 7.3: Comparison of two FP numbers with same signs.

sign of a	sign of b	abs	output
+	+	|a|>|b|	b
+	+	|a|<|b|	a
-	-	|a|>|b|	b
-	-	|a|>|b|	a

7.2.4 Inverse square root unit

RRT algorithm involves inverse square root calculation in new RRT points generation. The hardware resources/power consumption and time latency are too expensive to afford if it is tackled in frontal attack. An method based on Newton approximation arose in the source code of *Quake3* 3D game launched in 1990s presents an incredible solution to this intractable problem. Typically, Newton method of approximation iterates several times to get a high precision result. Miraculously, the *Quake3*'s method converges to a low error solution only after the first iteration. A magic hexadecimal constant '5f37_59df' plays a important role in this method. Incidentally, the author of this method is still in mystery [285]. A deduction of this approach is detailed in [286]. C codes of this method are given as follows.

```
1 float InvSqrt(float x)
2 {
3    float xhalf = 0.5f*x;
4    int i = *(int*)&x; // get bits for floating value
5    i = 0x5f3759df - (i>>1); // gives initial guess y0
6    x = *(float*)&i; // convert bits back to float
7    x = x*(1.5f-xhalf*x*x); // Newton step
8    return x;
9 }
```

Converting these codes into Verilog can be done in a ingenious way: use Verilog arithmetic operator to calculate line 5 and instantiate FP adder and multiplier to compute line 7 of the C codes. The underlying thought is that an FP number (no matter its format) is represented by a binary string. This string can be translated to an FP number but also to an integer, depends on the context. When two such strings are manipulated by Verilog arithmetic operators (or other HDL operators), they are regarded as binary strings so as the result. If the result is used as an operand in an FP unit, it is interpreted as a FP number. Consequently, the conversion from FP to integer or vice versa is not necessary in HDL.

Solving an arithmetical expression is essentially a serial process for the precedences of operators are different. Even for operators with the same priority, they should be executed in sequential. The intrinsic serialism of arithmetic is the origin why adders and multipliers are triggered at distinct time. To materialize *Quake3*'s method in FPGA, line 3 is carried out not by instantiating FP multiplier but with a subtraction of one from the exponent field of the input FP number x. This much more simple operation can achieve the same effect when dividing a FP number by 2 is needed. After line 4 completes, *Fp_mul_Ess* performs $x * x$. Due to *Fp_mul_Ess* cost one clock cycle to output its product, *Fp_mul_Es* begins to calculate $xhalf * x * x$ one cycle later. *Fp_add_Egsc* computes $1.5f - xhalf * x * x$ one cycle after *Fp_mul_Es*. Wait one more cycle, *Fp_mul_Esc3* starts to compute $x * (1.5f - xhalf * x * x)$.

In [286], another constant '5f37_5a86' having better precision is deduced. '5f37_5a86'is employed other than '5f37_59df' to design inverse square root module, although the increased precision achieved by the former constant is trivial.

For the difficulty of design high performance FP divider, the Xilinx FP IP core is used to do divisions instead of developing it from scratch. The latency of FP divider is customized to one clock cycle, in accordance with devised FP adder and multiplier. But this divider is triggered at rising edge of clock, we cannot change this because it is a packed IP core. To make divider triggers by flag generated by preceding unit, assign zero to dividend and one to divisor so the result is zero before the flag is one. When at the clock rising edge and flag has value one, assign actual dividend and divisor value to divider. By this way, the execution of divider is ordered.

7.2.5 Floating point random number

There are two types of random number generator which can be implemented in FPGA: true random number generator (TRNG) and pseudo random number generator (PRNG) [287, 288, 289]. In consideration of the difficulty to design a TRNG, this research designs a PRNG producing IEEE 754 FP number in the range of $(0, 1)$ basing on linear feedback shift register (LFSR). XOR is utilized as the operation among certain bits of the register to jumble up the order of pseudo random numbers. The penalty of LFSR based PRNG is that the sequence of generated pseudo number is invariant if the seed is constant. To address this problem, two LFSRs are concatenated so that the output of the first LFSR is input to the other as seeds, making the seed always in dynamic to augment randomness. The LFSR based PRNG is triggered only by clock rising edge so is always active.

RRT algorithm requires the random number range to be $[0, 1]$ instead of $(0, 1)$. Yet, as can be seen from Table 7.1, 0 is an exception FP number with unique bit representation. The exponent of any IEEE 754 real number in $(0, 1)$ is '0111_1110', while 1's exponent is '0111_1111' (its mantissa is all-zero). Thereby it is more pragmatic to sacrifices 0 and 1 because of their uniqueness. In the first attempt, it is tried to produce the 23-bit pseudo random number by giving a 23-bit number as the static seed for the first LFSR, and input the pseudo random number to the second LFSR as its dynamic seed. Then concatenate '0_0111_1110' to the output of the second LFSR to constitute a 32-bit FP number. However the pseudo random number produced tend to be very small. This method is modified by generating 27-bit pseudo random number, setting the exponent to '0111_1110' if it is larger, and concatenating '0_0111' to the 27-bit number. By this way, a more uniform distributed FP random number sequence is obtained.

7.3 GNPS arranged RRT register transfer level model design

Verilog is employed as the HDL to design RTL model of RRT-GNPS. The "module" hereafter refers to Verilog module. For the functions of program usability predicates, the computational process of a GNPS is deterministic. Meanwhile, the introduction of flag signals produced by edge detection sequences the processing order of RRT-GNPS. Under this circumstance, the function of predicates is substituted by flag signals. The root cause of this replacement is that we cannot instantiate a module (for example an FP unit) in the *if-else* construct of HDL while fixed point arithmetic expressions can be contained for the using of HDL arithmetic operators, where predicates of programs are expressed in the condition of *if*.

For arithmetic computations are performed by instantiating FP units correspondingly, the RTL model of RRT-GNPS illustrated in Figure **??** is designed in accordance with arithmetic operations in it. Figure 7.7 presents the RTL model generating one RRT point in an environment with two obstacle points. Due to the conspicuous structural complicity, it is unfeasible to present a legible panorama RTL model block diagram of RRT-GNPS so only this part is given (In effect, Vivado indeed draws an exhaustive schematic. Nevertheless, because of the sophistication, the connections are too small to read when expand modules to FP unit level). Robot initial point (x_1, y_1) is the root of all RRT point. When compute the first potential RRT point, it is the only RRT point and the nearest point to the first random point. So there is no comparison module to determine the nearest RRT point.

Two LFSR based PRNG modules `random` keep working all the time to produce random FP number sequence and their random number output ports are connected to module `co_rand1` which generates the first random point (x_{rand1}, y_{rand1}). The RTL model of RRT-GNPS begins to work after the port *begin* of `co_rand1` receiving a rising edge. This is done by connecting port *begin* to the pin of a button in the FPGA developing board. `co_rand1` contains two *Fp_mul_Eg* multipliers to execute rule Pr_{1-1} and Pr_{2-1} of RRT-GNPS concurrently. After random point (x_{rand1}, y_{rand1}) obtained, the *control* port of `co_rand1` emits a flag signal to the *state* port of module `d1_1` to stimulate it, which begins to calculate the square distance between (x_1, y_1) and (x_{rand1}, y_{rand1}) (Pr_{4-1}). `d1_1` is composed of two *Fp_add_Egs* which compute the coordinate differences between initial point and the random point, two *Fp_mul_Es* that calculates the squares of two differences, and one *Fp_add_Egsc* gives the square sum.

Module `Fp_invsqrt1` calculates inverse square root of d_1^1. Please look back at Subsection 7.2.4 for the details of the structure and function of this unit. Alongside the processing of `Fp_invsqrt1`, compute coordinate difference of (x_{rand1}, y_{rand1}) and (x_1, y_1) with two *Fp_add_Egs*. Then flag signal from `Fp_invsqrt1` activates module `coor_new1` comprising four *Fp_mul_Ess* and two *Fp_add_Egsc* to calculate the first potential RRT point (x_{new1}, y_{new1}) (Pr_{6-1} and Pr_{7-1}).

After the potential RRT point derived, if the line segment taking (x_1, y_1) and (x_{new1}, y_{new1}) as the two endpoints is obstacle free, i.e., all the distances from obstacle points to this segment are larger than robot rotation radius (ξ), then this potential point is indeed a RRT point. This verification begins with the simultaneous computation of Pr_{1-2}, Pr_{2-2}, Pr_{3-2} and Pr_{4-2} by module `dif_mul_sum`, `d_1`, `d_2` and `d_3` respectively. Module `coor_p1` deals with program Pr_{8-2} and Pr_{9-2}. Pr_{11-2} is coped with `d_4`. The distance selection program Pr_{13-2} is determined by `Comp`. All these modules are packed in a higher level module `pdist`.

It is highlighted that distances between eight obstacle points and the line segment $(x_1, y_1) - (x_{new1}, y_{new1})$ are computed in parallel so the performance will be improved to a large extend, especially for large amount of obstacle points. When the eight distances obtained, seven `Min_sq`s in total are needed to figure out the smallest distance. This is because `Min_sq` can only compares two FP numbers at a time. To find out the minimal value from eight distances, at first four `Min_sq` taking in the eight distances can pick out the larger four. Another two `Min_sq`s determine the larger two among four distances. Execute $\xi - distance$ to the two smaller distances and utilize another `Min_sq` to output the minimal distance. If it is negative, the segment is obstacle free and the potential point is a valid RRT point. Then the first RRT point is stored as (x_{rrt1}, y_{rrt1}), sending a flag signal from port *control2* of `Min_sq` to activate the computation of the second potential RRT point. Otherwise discard this point, transmit a flag signal from port *control1* of `Min_sq` to port *state* of

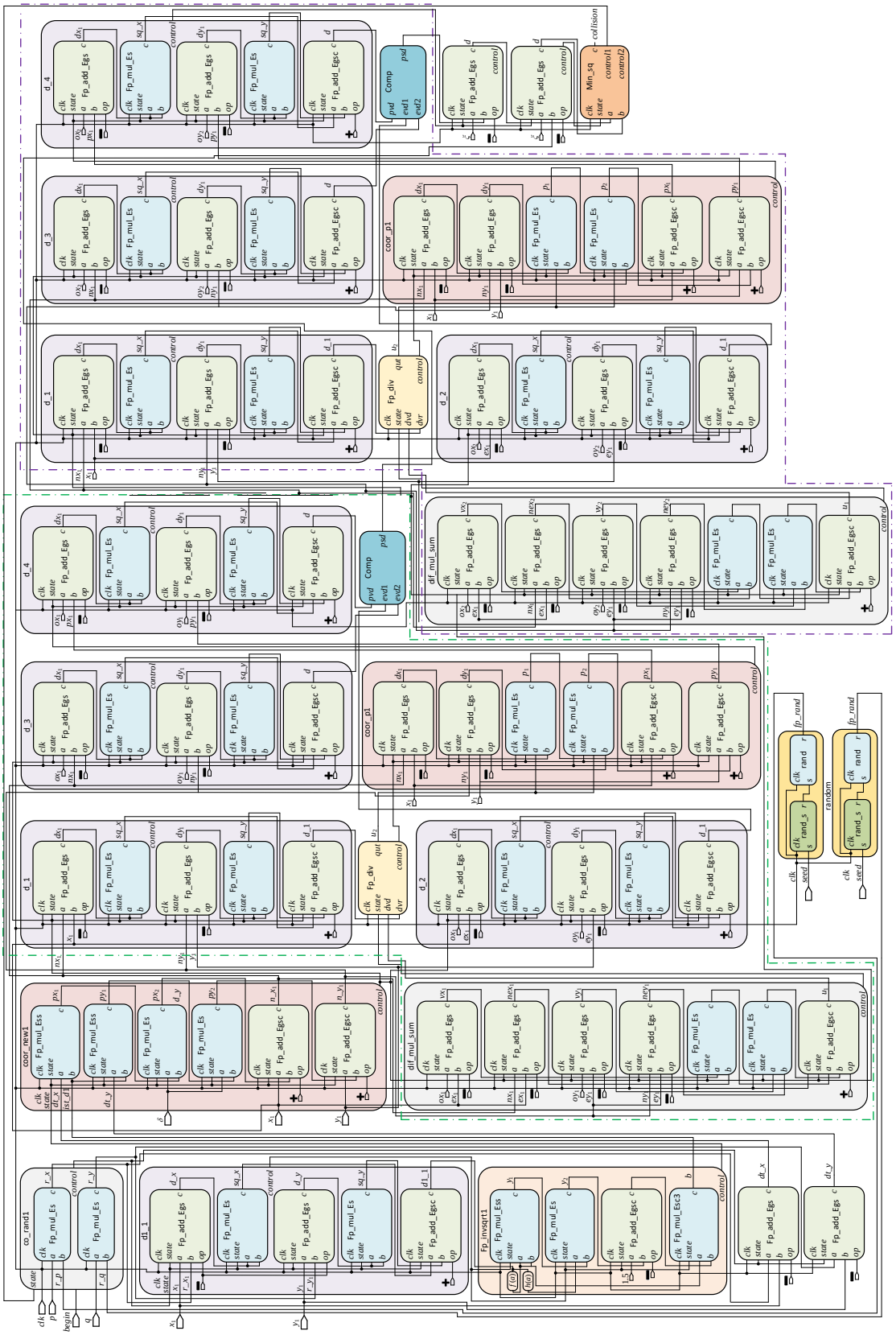


FIGURE 7.7: RTL model of the RRT-GNPS generating the first RRT point.

`co_rand1` to resume the calculation of (x_{new1}, y_{new1}) until it is a RRT point. Membranes included in two dashed blocks in Figure **??** carries out the work of the seven `Min_sq`s.

In order to compute the second potential RRT point, the nearest point to the second random point (x_{rand2}, y_{rand2}) should be chosen from (x_1, y_1) and (x_{rrt1}, y_{rrt1}). This process corresponds to rules Pr_{1-18} to Pr_{8-18} in RRT-GNPS and module `nearest2` performs this work. After the second potential point (x_{new2}, y_{new2}) obtained, carry out the verification stated above to confirm whether it is a RRT point. An unified modeling language (UML) activity diagram depicted in Figure 7.8 details the whole process of RTL model.

To scale the module to incorporate more obstacle points, one can instantiate more `pdist` modules and enlarge the comparison logic for selecting the minimum distance.

7.4 FPGA implementation of RRT-GNPS

The hardware facilities involved in FPGA implementation are a host computer equipped with a Intel Core i7-7820HQ and 16 GB RAM, a Xilinx VC707 evaluation board featured a Virtex-7 xc7vx485t-2ffg1761 FPGA [290]. The FPGA integrated developing environment employed is Xilinx Vivado 2019.1.

For the sake of ensuring RTL model behaves as expected, a testbench should be designed to validate it. A RRT-GNPS RTL model is instantiated in the testbench, together with the clock cycle declaration and initial value setup of input variables. Then perform RTL model behavioral simulation which presents the model behavior by drawing waveforms of variables. The RTL model turned out to function well if waveforms and values meet design objectives. Set clock period as 10 ns, the behavioral simulation waveform of RRT-GNPS is given in Figure 7.9. As can be seen, the first two RRT points obtained are coincident with the two in Figure 7.3. Note, the length of clock period is not important for behavioral simulation because it is a software simulation conducted by host computer CPU and all the gate latencies and datapath latencies are neglected. As a consequence, behavioral simulation can verify the functionality of a RTL model but void of timing analysis.

If the behavior of RTL model meets requirements, the subsequent action is to synthesize design. Once synthesis completing, perform constraints and hardware debug cores setting. Obstacle points are assigned to corresponding to variables via Verilog system task "$readmemh" from a .txt file stored in host computer, instead of accessing these data through pins. So input ports of RRT-GNPS RTL model are clock input and `co_rand1`'s *begin* port. The computational results of FPGA cannot be observed directly but can be checked by performing hardware debug in the integrated logic analyzer (ILA). To observe a variable, a debug core should be set in the synthesized model to probe values obtained in FPGA. Perform debug core setting after synthesizing model at least once.

Target FPGA has two differential clock pins. To generate a clock form these two pins, a global clock input buffer IBUFGDS whose input ports are the two pins should be instantiated to output a clock with period defined by timing constraints. The second RRT point's y-coordinate (y_3) is selected to output so pins should be allocated to y_3 port. This is a vector port with 32-bit hence it needs 32pins. One matter should be pointed out, however, that two pins AM31 and AG32 in bank 14 of target FPGA cannot be used for they are involved in the JATG circuit. If they are assigned to other ports, the JTAG will be disconnected and programming device by JTAG is not

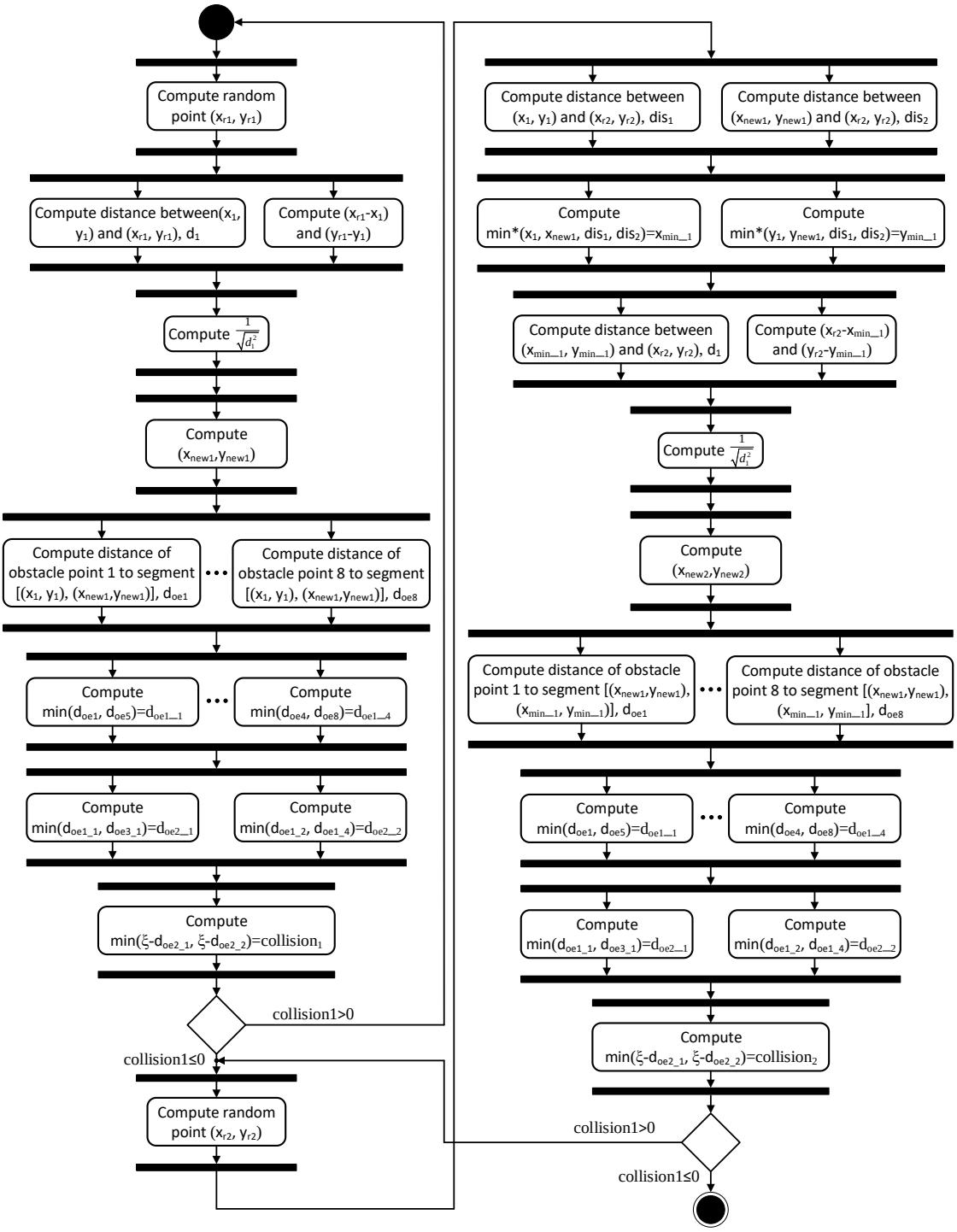


FIGURE 7.8: The whole process of RRT-GNPS RTL model characterized by UML activity diagram.

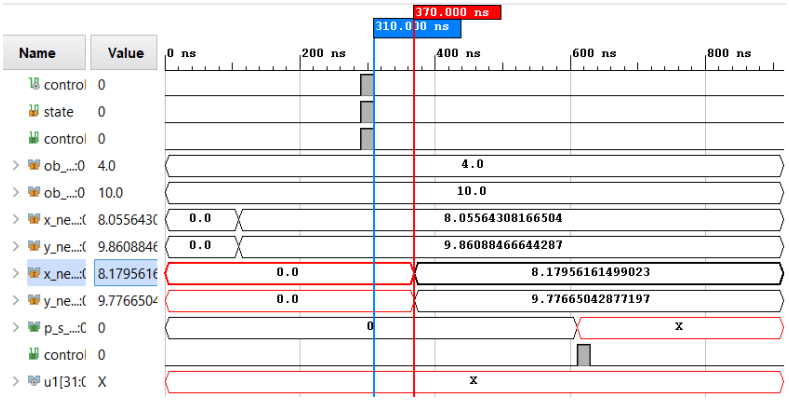


FIGURE 7.9: The behavioral simulation waveform of RRT-GNPS RTL model. The red waveforms are caused by the lack of initial values of variables. This phenomenon can be eliminated by assigning initial values to *reg* variables.

able to work. This fact is not mentioned in VC707 evaluation board user guide so it is emphasized here. The physical constraints of this design is shown in Figure 7.10. With respect to the complicity of RRT-GNPS, the clock period is set to 40 ns, which means the FPGA hardened RRT-GNPS computes 2.5×10^7 times per second (the frequency is 25MHz). This set may be modified after implementation if it fails to meet timing closure.

After RTL model is synthesized successfully, carry on implementation in which conducts netlist/power optimization, reports timing together with hardware resource/power consumption, and completes *Place* & *Route*. Post implementation timing simulation of RRT-GNPS is shown in Figure 7.11. According to static timing analysis results, the worst negative slack (WNS) is 2.518 ns, while worst hold slack (WHS) is 0.048 ns. Therefore the 40 ns clock period can meet timing constraints. FPGA costs nearly 3059.01 ns to obtain two RRT points while host computer CPU costs 0.097948 s to get them. So the speedup is computed in Formula 7.1. Hardware debug of PGA hardened RRT-GNPS is shown in Figure 7.12.

$$9.7948 \times 10^7 \div 3.05901 \times 10^3 = 3.20195 \times 10^4 \tag{7.1}$$

The resource utilization and power consumption of RRT-GNPS are shown in Figure B.10 in Appendix B. RRT-GNPS costs 43% of look-up table (LUT) because of the long bit width of IEEE 754 representation. If narrow bit width FP format. e.g., 16-bit FP, or fixed point representation is used, the LUT utilization can be reduced by almost half. 16% of DSP resource is utilized by part of arithmetic. Narrow bit width FP or fixed point format will decrease this amount as well. The consumptions of other resources make up small percentages. The *Place* & *Route* of RRT-GNPS is given in Figure B.11.

7.5 Wrap-up

The RRT algorithm is modeled in the framework of GNPS in which parallel procedures are executed simultaneously to speedup the operation process. IEEE 754 float point format is employed to represent variables to provide large dynamic range and high computational accuracy. Float point adder and multiplier are designed from

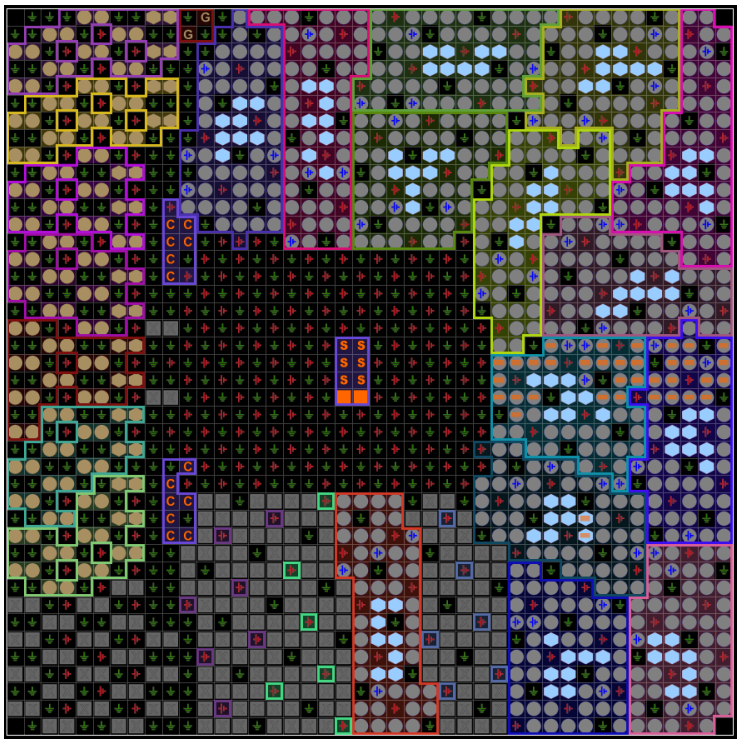
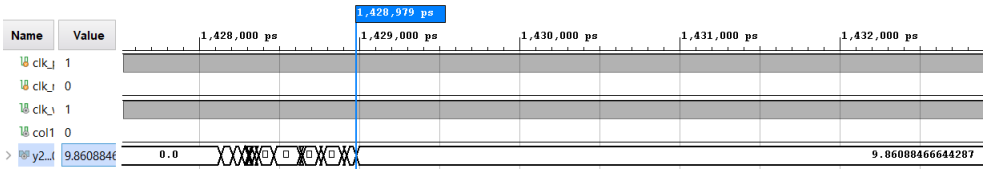
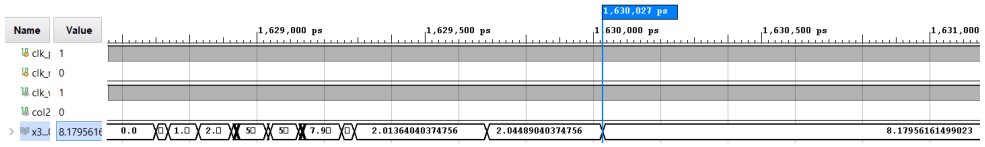


FIGURE 7.10: The physical constraints of RRT-GNPS shown in the package view of FPGA. Pins with orange bars inside signify the occupied pins. The light blue hexagon pins are clock pins while the gray round pins are ordinary input/output pins.

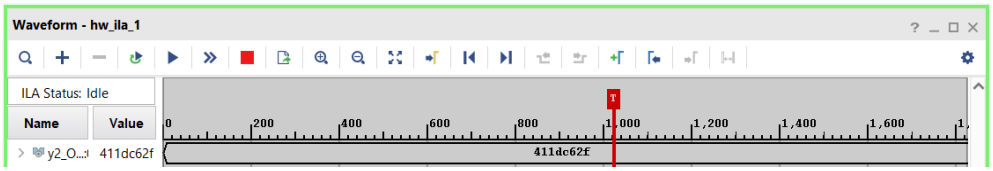


(a) Post implementation timing simulation waveform of (x_2, y_2).

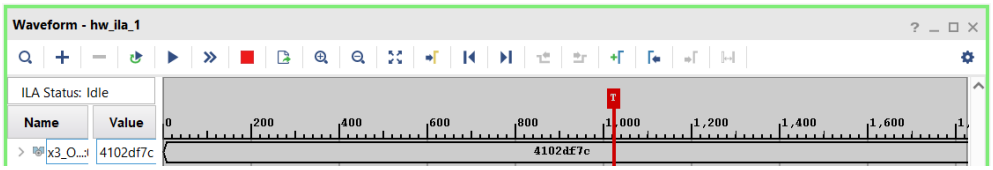


(b) Post implementation timing simulation waveform of (x_3, y_3).

FIGURE 7.11: The stable value of first RRT point (x_2, y_2) appears at 1428979 ps (1428.979 ns) while the second RRT point (x_3, y_3)arises at 1630027 ps (1630.027 ns). So it costs $1428.979 + 1630.027 = 3059.01$ ns to get results.



(a) Hardware debug of y_2 (y coordinate of the first RRT point).



(b) Hardware debug of x_3 (x coordinate of the second RRT point).

FIGURE 7.12: Hardware debug of RRT-GNPS. Numbers are represented in hexadecimal. As can be validated, 411dc62f is the IEEE 754 format of decimal number 9.86088466644287. This result coincides with post implementation timing simulation result given in Figure 7.11.

scratch thus control nodes can be planted inside to improve performance. The float point inverse square root unit is constructed based on the adder and multiplier so that this unit refrains from division operations, which are hardware expensive and performance low. A special pseudo random number generator outputting IEEE 754 complaint float point random numbers in the range of $(0, 1)$ is devised on the basis of linear feedback shift registers. For those divisions cannot be avoided, Xilinx float number operation IP cores are instantiated to perform divisions for their high speed. Contrast to software simulation, the speedup of FPGA implementation achieved is 3.20195×10^4.

Chapter 8

Conclusions

Hardware implementation of bio-inspired computing models is a research line that focuses on realizing the massive parallelism using digital circuits that is impossible to do with a software program. This thesis targets two bio-inspired models, reaction systems and numerical P systems, and investigates how to implement them on FPGA re-configurable hardware, and their applications in different realms.

The introduced normal form of reaction systems in which reactions have only one entity as products and elements of product set are disjoint with entities in the context sequence, it is not difficult to construct a sequential switching circuit, which is a Mealy or Moore machine processing 1-bit variables. On the other side, from the Mealy or Moore machine of a synchronous circuit, one can derive Boolean equations demonstrating relationships among input, output and internal states. Binary states of variables in the right hand side of these Boolean equations can be regarded as positive or negative literals which are transformed to reactants and inhibitors. Internal states and output in the left hand side of Boolean equations are deemed as products. So synchronous circuits can be converted to reaction systems.

Threshold supply assumption of RS is implemented as "presence 1 absence 0" to reflect the qualitative feature. This treatment is expedient for designing a RS carrying out some biological process and for its FPGA implementation, which is validated by the implementation of self assembly intermediate filaments RS and heat shock response RS. On the contrary, "1 presence 0 absence" method is suitable for devising a RS counter and its FPGA implementation. These two treatments coincide with the relationship of RS and synchronous circuits, namely, a RS is essentially a Mealy or Moore machine which corresponds to a synchronous circuit. The duration of an entity (one step in the interactive process) is realized as one clock cycle to meet the no permanency assumption. Consequently, an integral FPGA implementation method of RS is proposed and tested.

Computation of applicable rule set of symbol object P systems is tackled from the integer multi-criteria linear optimization point of view. In the *max* derivation mode, all elements of the set of applicable rule multiset are solutions of a system of inequalities defining a Diophantine equations. By constructing "either-or" constraints in integer linear programming, it is possible to solve Diophantine sets more readily. New constraints introduced make sure that any solution of the inequalities is Pareto-optimal, which corresponds to a maximally parallel rule multiset.

The relationship of NPS and system of difference equations is revealed: NPS is another form of systems of difference equations. These two forms can transformed to each other. Their main difference consists in that programs of NPS give the recurrence relations among variables while equations in the difference system tell variable values varying with the discrete time t. Generally, the physical meanings of the model designed by NPS are more transparent. Constructing a NPS from a given difference system is more straightforward. Difference systems can be transformed

from differential systems via standard methods, so differential systems can be approximated by NPSs when simulate these differential systems on discrete systems such as a computer. This means that NPSs can model differential systems which are widely used in engineering filed. Moreover, it can be observed that the notion of NPS corresponds to Mealy/Moore machine operating with numbers instead of bit values.

The symbol object P system can be converted to a special NPS because a rule can be transcribed as a set of programs in which each program updates multiplicity of one type of object. In every membrane, consolidating all programs updating the same object, the number of programs is reduced. These more complicated programs have the same effect as object rewriting rules. This research tries to look at symbol object P systems and NPSs in an uniform point of view, and more importantly, to use an intelligible manifestation to transcribe rewriting rules. This transcription of rules enlightens the FPGA implementation of GNPS programs.

Have in mind the discrepancies of enzyme catalyzed programs of ENPS and the limitation of the only predicate form ($e > max(v_1, \ldots, v_n)$ or $e \geq max(v_1, \ldots, v_n)$), the definition of NPS is extended to generalized numerical P system (GNPS) to facilitate FPGA implementation and to enlarge application scope of NPS by introducing Presburger arithmetic predicates. Variables of GNPS are classified as input, output and internal variables for that RTL models of GNPS have input and output ports to be mapped to FPGA input and output pins receiving data and transmitting computation results.

The core issue of FPGA implementation of GNPS lies in how to implement programs. A method entitled as *FPGA Step-wise Parallel Implementation of Program* (FSPIP) is devised for this issue. The FSPIP has three steps: the first step computes all production functions of applicable programs, the second step calculates repartition protocols and the third step sums repartitioned values of one type variable and assigns new values. The third step inspired from the relation of rules and programs avoids the conflict arising from multiple programs updating the value of the same variable. Since GNPS is the superset of NPS and ENPS, FPGA implementation method of GNPS is applicable for the other two.

Several robot controllers based on NPS, ENPS and GNPS are implemented in FPGA to validate FSPIP method. These models work with fixed point number representation. So real numbers are amplified to 2^n (n is the number of bits for fraction part of a real number) times larger and rounded to the nearest integer. After computation, results are large integer numbers which are shrunk to corresponding real numbers by left shift n bits. A GNPS working on IEEE 754 floating point numbers and executing RRT algorithm is also designed. Arithmetic operations of floating point numbers are carried out by instantiating devised FP arithmetic units, such as adder, multiplier and inverse square root unit. This downside of Verilog HDL disables the normal treatment of predicates which are interpreted as logical expression in Verilog *always* blocks. A signal edge detection based method is adopted to fire FP arithmetic units in accordance with priorities of arithmetic operations and to trigger the predetermined sequence of other operations which are defined by predicates of programs.

The major achievements of this thesis are the FPGA implementation methods of RS and GNPS. FPGA implementation of RS opens a novel research line to speedup large scale RS and other biological models, in the perspective of digital circuits and parallel architectures. This study presents the tight relations between synchronous circuits and qualitative models, and proposes a natural and effective way to address

qualitative features. It proves that not only quantitative models, but also qualitative models can be implemented in FPGA efficiently.

Many kinds of P systems, like tissue-like P systems, population dynamic P systems and spiking neural P systems have been simulated or implemented in CPU and GPU, while only symbol object cell-like P systems were implemented in FPGA before the work in this thesis. Numerical P systems belong to cell-like P system category, but they are quite different from symbol object P systems. The FPGA implementation method of GNPS working on fixed point number and floating point number paves the way to leverage the maximal parallelism of P systems to accelerate computations in engineering field, especially in consideration of the fact that most engineering models are modeled by differential systems. The devised method can also provide a good reference to implement other types of P systems on FPGA.

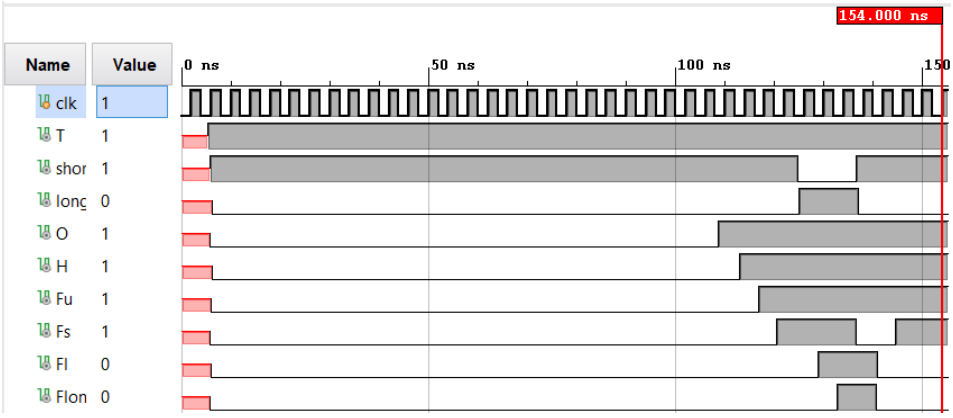
The realized speedups of FPGA hardened GNPS show that FPGA based implementations of GNPS are quite suitable for computation-intensive algorithms. Many artificial intelligence algorithms are hungry for computing powers. Arrange some of these algorithms in the framework of GNPS and implement them on FPGA is profitable in terms of the noticeable speedups. Image and video processing is a apposite domain to exploit the advantage of GNPS and its FPGA implementation. GNPS FPGA implementation can also be put to good use in applications requiring high real time performances, for example natural language processing, face and gait recognition and real time control.

The huge success of deep learning energizes the development of bio-inspired computing. Spiking neural P systems (SNPs) model neurons and spikes among them. Investigation of hardware implementation of SNP would be quite promising taking into account the potential to build a deep learning network on the basis of SNP. The extensive of applications of deep learning would promote the progress of SNP and related P systems.

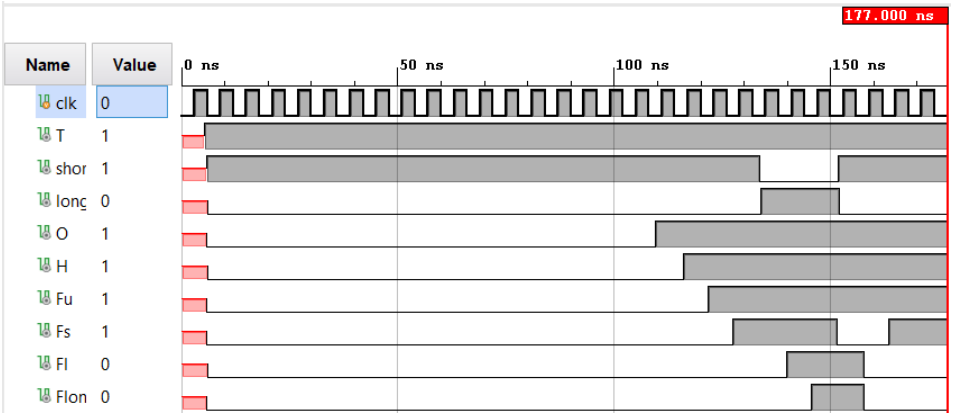
Playing with FPGA is a tedious task requiring professional digital circuits design knowledge. A compiler from models to be implemented in FPGA to their HDL codes will simplify the design process substantially. In this thesis, conceiving a GNPS and then compiling it to Verilog codes shows that such a design method is viable and friendly for software developers. On the other side, imagine that all hardware resources are employed but still not sufficient to process a large amount of data. If the data is divided into several parts and input them sequentially, then it would be a good solution. So for FPGA implementation and its application, what still lacks are relevant software which can (partially) automate design process and organize data in different portions, then import data in a way that makes full use of computing nodes implemented in FPGA. On the whole, only FPGA hardware cannot exert its benefit. Hardware plus software would be more reasonable.

Appendix A

Associated figures of reaction systems

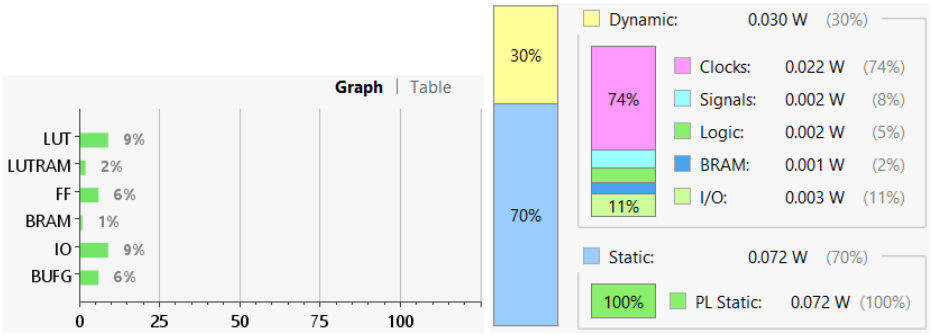


(a) Post implementation timing simulation with 4 ns clock period.



(b) Post implementation timing simulation with 6 ns clock period.

FIGURE A.1: Post implementation timing simulation of $\mathcal{R}_{ifl2}$.

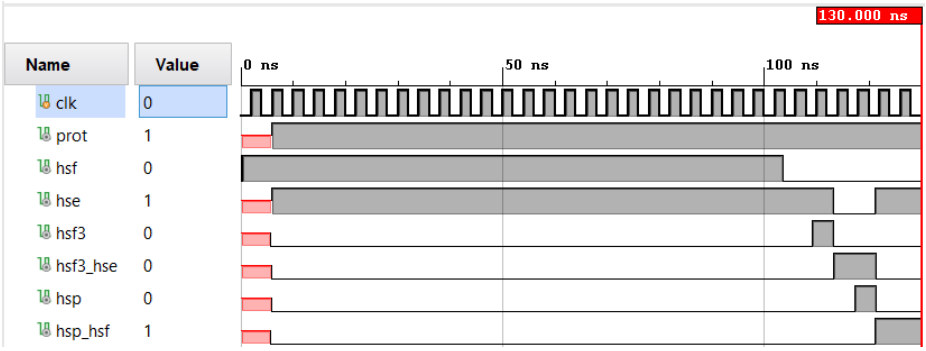


(a) Hardware resources dissipation. (b) power consumption.

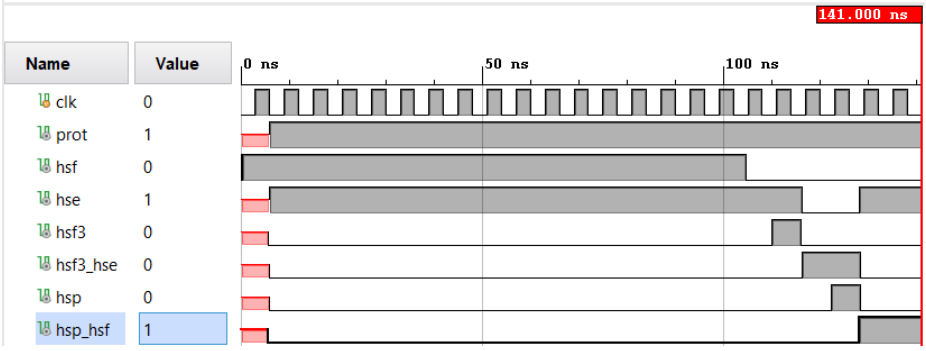
FIGURE A.2: Hardware resources dissipation and power consumption of $\mathcal{R}_{ifl2}$.

(a) I/O planning of $\mathcal{R}_{hsr1}$. (b) I/O planning of $\mathcal{R}_{hsr2}$.

FIGURE A.3: Physical constraints settings of two heat shock response reaction systems. Gray circles with wide orange bars are pins allocated to RTL model ports.

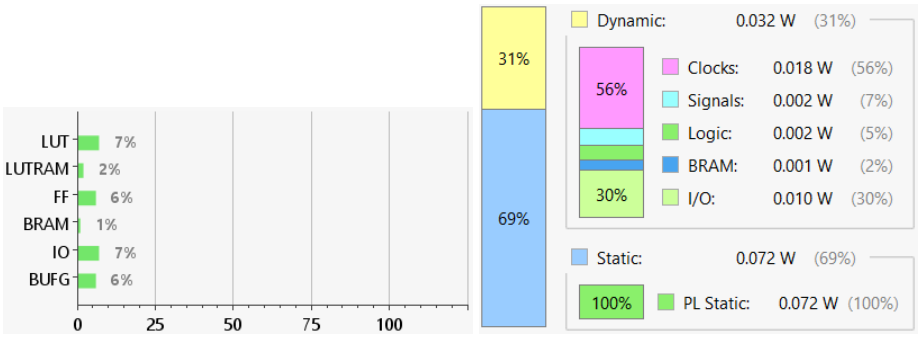


(a) Post implementation timing simulation with 4 ns clock period.



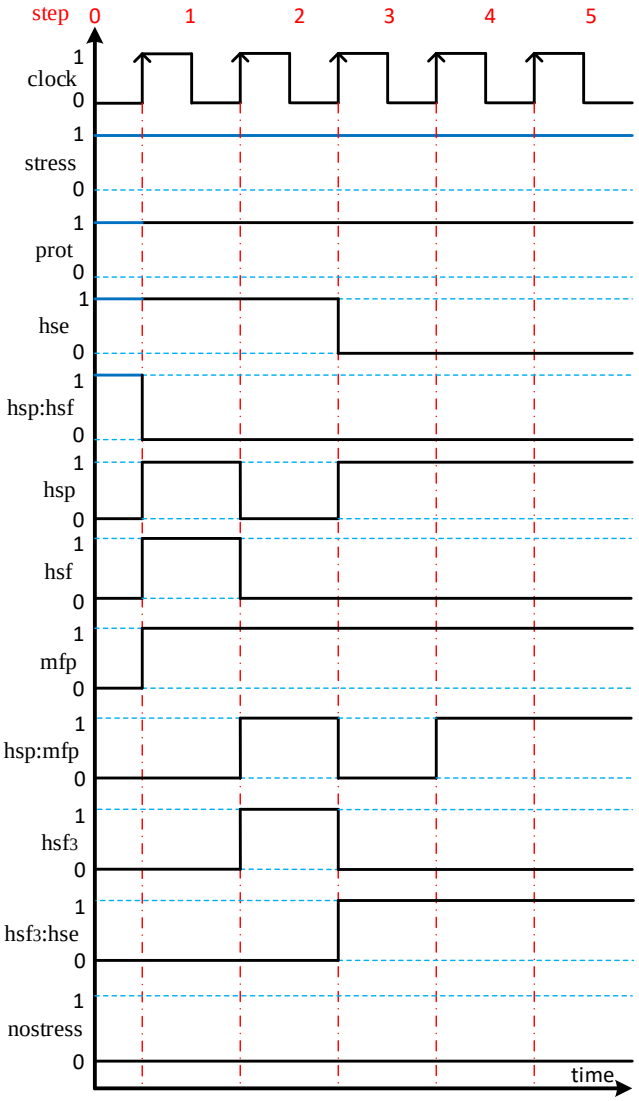
(b) Post implementation timing simulation with 6 ns clock period.

FIGURE A.4: Post implementation timing simulation of $\mathcal{R}_{hsr1}$.

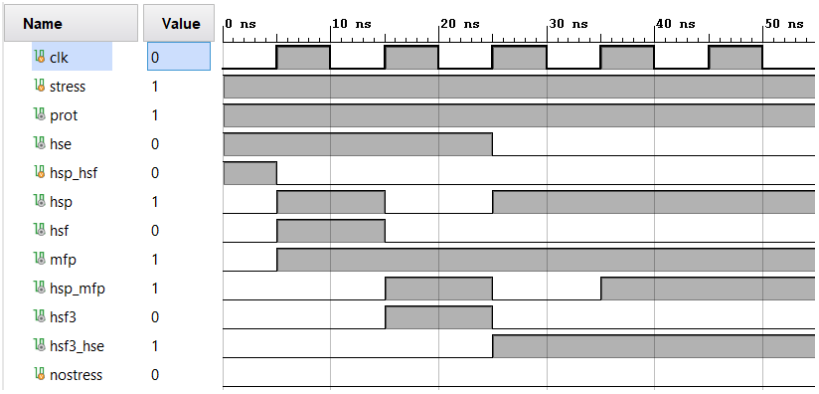


(a) Hardware resources dissipation. (b) power consumption.

FIGURE A.5: Hardware resources dissipation and power consumption of $\mathcal{R}_{hsr1}$.



(a) The expected waveform of interactive process listed in Table 4.9. The blue solid lines indicate context entities.



(b) Vivado behavioral simulation waveform of RTL model. The clock perid is set as 10 ns.

FIGURE A.6: Waveform comparison of interactive process 2 of $\mathcal{R}_{hsr} = (B_3, A_3)$.

Appendix B

Associated figures of GNPS

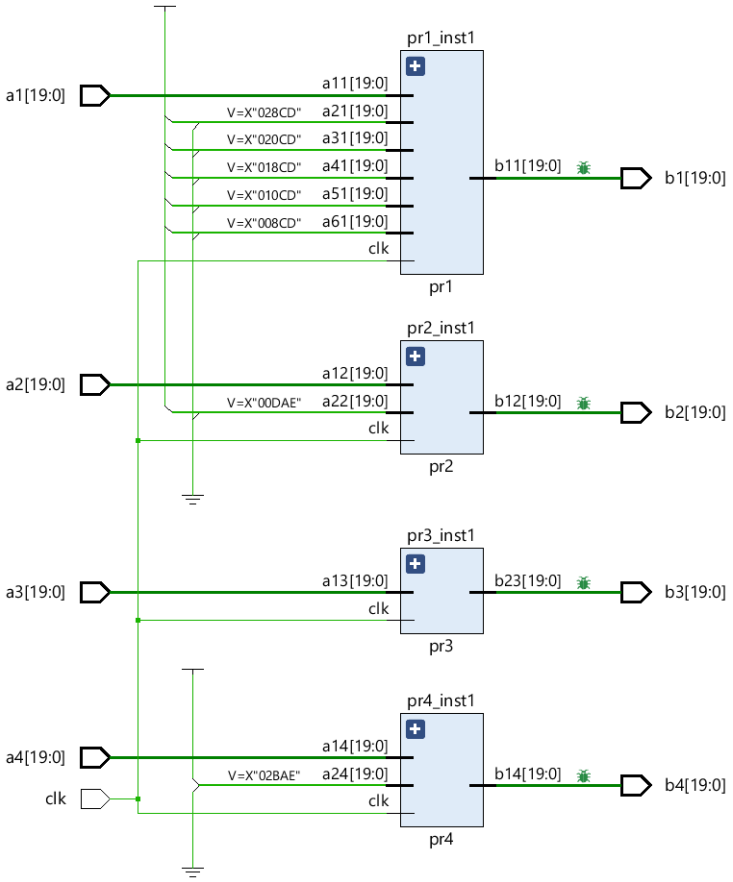
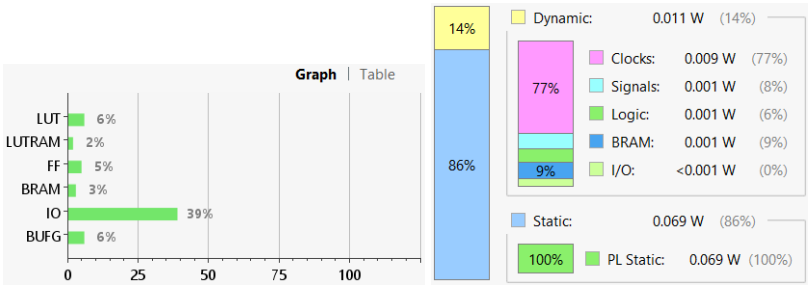


FIGURE B.1: Block diagram of GNPS1 RTL model. After the design of a model, the corresponding schematic can be drawn automatically by Vivado. The schematic characterizes the same functions/behaviors as RTL model representing by HDL.



(a) Hardware resources utilization of GNPS1.

(b) Power consumption of GNPS1 is 0.08 w.

FIGURE B.2: Total power consumption is the sum of device static power and dynamic power. Power consumption of the two cases are nearly the same, although GNPS1 works in all parallel and GNPS2 works in sequential.

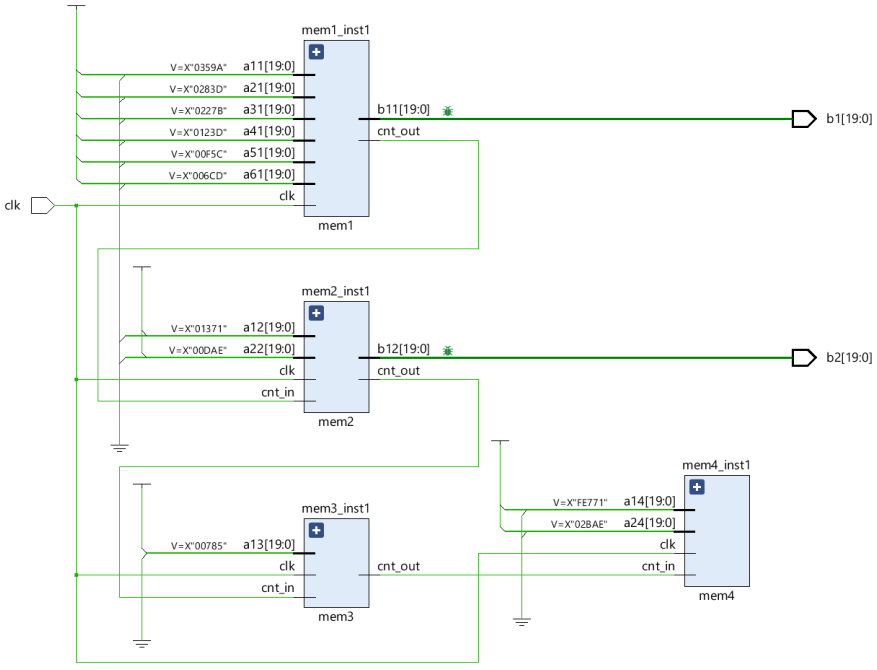
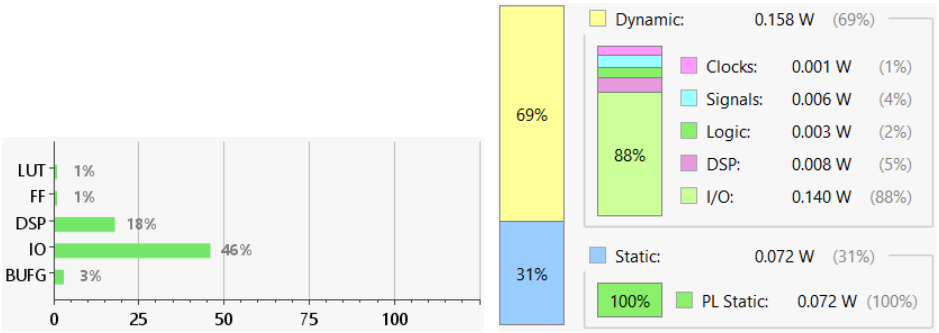


FIGURE B.3: Block diagram of GNPS2 RTL model. Each membrane is modeled in Verilog basic functional unit, *module*. The bug icons indicate variables to be debugged in *Hardware Debug* procedure.

FIGURE B.4: I/O planning of NPS1 RTL model. NPS1 has one 1-bit input port *clock* and two 24-bit output ports *rw* and *rw*, so totally 49 pins are distributed to these three ports.



(a) Hardware resource dissipation of NPS1. (b) Power consumption of NPS1 is 0.23 W.

FIGURE B.5: Hardware resource dissipation and power consumption of hardened NPS1.

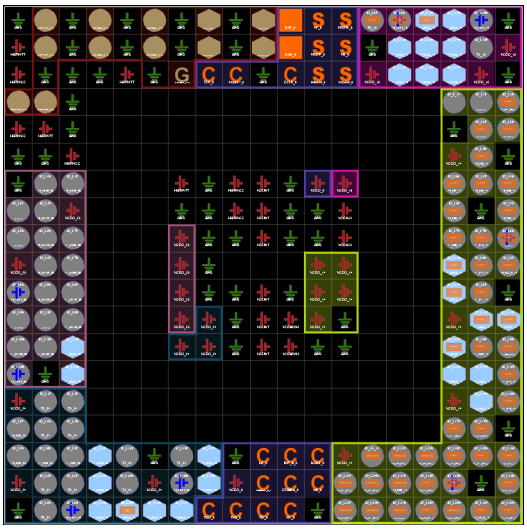
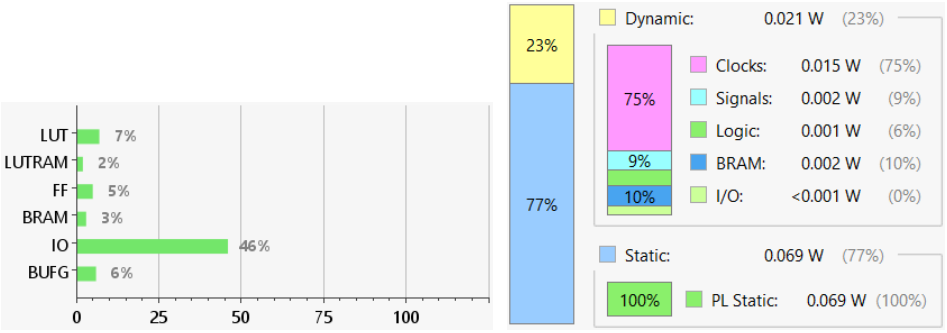


FIGURE B.6: I/O planning of ENPS1. 49 pins are used to represent I/O ports.



(a) Hardware resource dissipation of ENPS1. (b) Power consumption of ENPS1.

FIGURE B.7: Hardware resource dissipation and power consumption of hardened ENPS1.

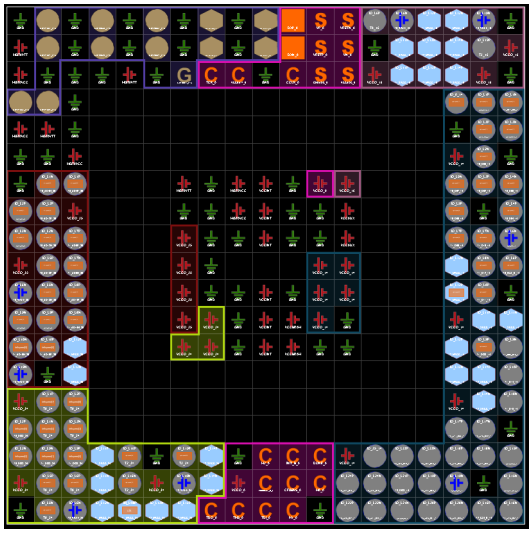
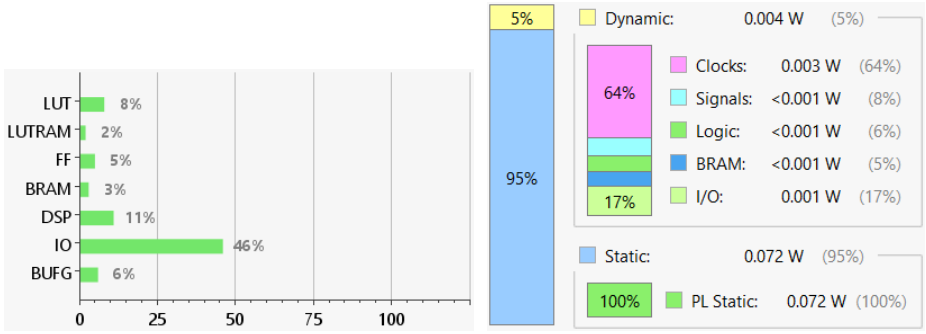
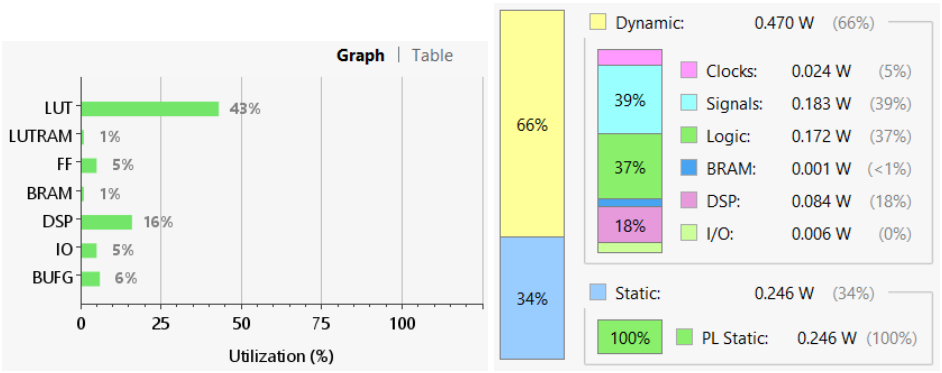


FIGURE B.8: I/O planning of GNPS3.



(a) Hardware resource dissipation of GNPS3. (b) Power consumption of GNPS3 is 0.076 W.

FIGURE B.9: Hardware resource dissipation and power consumption of hardened GNPS3.



(a) Hardware resource utilization of RRT-GNPS.

(b) Power consumption of RRT-GNPS is 0.716 w.

FIGURE B.10: The resource utilization and power consumption of RRT-GNPS.

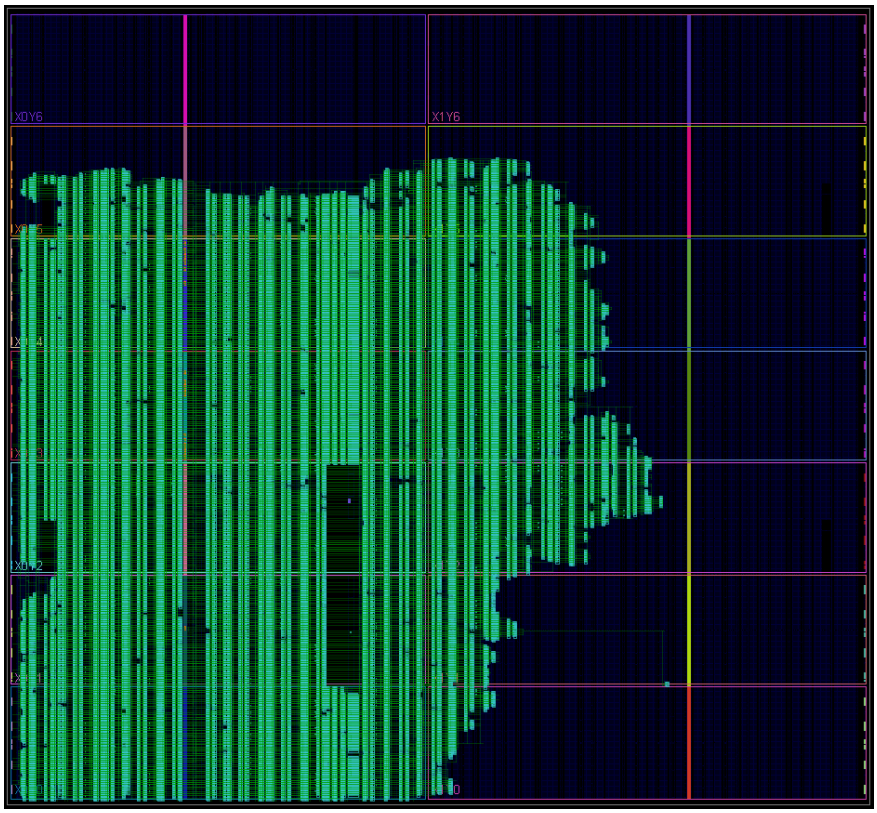


FIGURE B.11: The *Place* & *Route* of RRT-GNPS.

Bibliography

[1] L. Kari and G. Rozenberg. "The many facets of natural computing". In: *Communications of the ACM* 51.10 (2008), pp. 72–83 (cit. on p. 1).

[2] G. Rozenberg, T. Bäck, and J. N. Kok. *Handbook of natural computing*. Springer, 2012 (cit. on p. 1).

[3] H. J. Morowitz. *Beginnings of Cellular Life: Metabolism Recapitulates Biogenesis*. Yale University Press, 1993 (cit. on p. 1).

[4] J. P. Schrum, T. F. Zhu, and J. W. Szostak. "The origins of cellular life". In: *Cold Spring Harbor perspectives in biology* (May 19, 2010). DOI: 10.1101/cshperspect.a002212 (cit. on p. 1).

[5] S. L. Millerl. "A production of amino acids under possible primitive earth conditions". In: *Science* 117.3046 (May 15, 1953), pp. 528–529 (cit. on p. 1).

[6] S. L. Millerl. "Production of some organic compounds under possible primitive earth conditions1". In: *Journal of the American Chemical Society* 77.9 (May 12, 1955), pp. 2351–2361 (cit. on p. 1).

[7] K. Nakamura and N. Go. "Function and molecular evolution of multicopper blue proteins". In: *Cellular and Molecular Life Sciences CMLS* 62.18 (Aug. 9, 2005), pp. 2050–2066. DOI: https://doi.org/10.1007/s00018-004-5076-x (cit. on p. 1).

[8] M. Á. Medina. "Systems biology for molecular life sciences and its impact in biomedicine". In: *Cellular and Molecular Life Sciences* 70.6 (Aug. 19, 2012), pp. 1035–1053. DOI: 10.1007/s00018-012-1109-z (cit. on p. 1).

[9] P. G. Higgs. "Chemical evolution and the evolutionary definition of life". In: *Journal of Molecular Evolution* 84.5-6 (2017), pp. 225–235 (cit. on p. 1).

[10] E. Schr´odinger. *What is life?: With mind and matter and autobiographical sketches*. 2012. URL: http://strangebeautiful.com/other-texts/schrodinger-what-is-life-mind-matter-auto-sketches.pdf (cit. on p. 1).

[11] M. C. Weiss et al. "The physiology and habitat of the last universal common ancestor". In: *Nature Microbiology* 1.9 (July 25, 2016), pp. 1–8. DOI: https://doi.org/10.1038/nmicrobiol.2016.116 (cit. on p. 1).

[12] H. Tang, P. Thomas, and H. Xia. *Reconstruction of the evolutionary history of gene gains and losses since the last universal common ancestor*. arXiv. Feb. 16, 2018. URL: https://arxiv.org/abs/1802.06035 (cit. on p. 1).

[13] M. D. Cantine and G. P. Fournier. "Environmental adaptation from the origin of life to the last universal common ancestor". In: *Origins of Life and Evolution of Biospheres* 48.1 (July 6, 2017), pp. 35–54 (cit. on p. 1).

[14] L. P. Villarreal and V. R. DeFilippis. "A hypothesis for DNA viruses as the origin of eukaryotic replication proteins". In: *Journal of Virology* 74.15 (Aug. 2000), pp. 7079–7084 (cit. on pp. 1, 4).

[15] A. Dupressoir, C. Lavialle, and T. Heidmann. "From ancestral infectious retroviruses to bona fide cellular genes: role of the captured syncytins in placentation". In: *Placenta* 33.9 (Sept. 2012), pp. 663–671 (cit. on p. 1).

[16] M. S. Dodd et al. "Evidence for early life in Earth's oldest hydrothermal vent precipitates". In: *Nature* 543.7643 (2017), pp. 60–64 (cit. on pp. 2, 4).

[17] D. R. Wessner. "The origins of viruses". In: *Nature Education* 3.9 (2010), p. 37 (cit. on p. 2).

[18] H. P. Fischer. "Mathematical Modeling of Complex Biological Systems". In: *Alcohol Abuse and Alcoholism* 31.1 (2008), pp. 49–59 (cit. on p. 2).

[19] S. Motta and F. Pappalardo. "Mathematical modeling of biological systems". In: *Briefings in Bioinformatics* 14.4 (Oct. 2012), pp. 411–422. ISSN: 1467-5463. DOI: 10.1093/bib/bbs061. eprint: https://academic.oup.com/bib/article-pdf/14/4/411/479256/bbs061.pdf. URL: https://doi.org/10.1093/bib/bbs061 (cit. on p. 2).

[20] Z. Ji et al. "Mathematical and Computational Modeling in Complex Biological Systems". In: *BioMed Research International* 2017 (Jan. 2017), pp. 1–16. DOI: 10.1155/2017/5958321 (cit. on p. 2).

[21] J. Gunawardena. "Models in Biology: Accurate Descriptions of Our Pathetic Thinking". In: *BMC biology* 12 (Apr. 2014), p. 29. DOI: 10.1186/1741-7007-12-29 (cit. on p. 3).

[22] S. J. Singer and G. L. Nicolson. "The fluid mosaic model of the structure of cell membranes". In: *Science* 175.4023 (1972), pp. 720–731 (cit. on p. 3).

[23] N. L. Brun, S. T. Favorecido, and E. I. Sánchez. *Atlas of plant and animal histology–The Cell*. 2020. URL: https://mmegias.webs.uvigo.es/02-english/5-celulas/1-introduccion.php (cit. on p. 3).

[24] A. Ehrenfeucht and G. Rozenberg. "Basic notions of reaction systems". In: *International Conference on Developments in Language Theory*. Springer. 2004, pp. 27–29 (cit. on pp. 3, 13).

[25] A. Ehrenfeucht and G. Rozenberg. "Reaction systems". In: *Fundamenta informaticae* 75.1-4 (2007), pp. 263–280 (cit. on pp. 3, 13, 14, 58, 71).

[26] G. Rozenberg and J. Engelfriet. "Elementary net systems". In: *Advanced Course on Petri Nets*. Springer. 1996, pp. 12–121 (cit. on p. 3).

[27] A. Ehrenfeucht and G. Rozenberg. "Events and modules in reaction systems". In: *Theoretical Computer Science* 376.1-2 (2007), pp. 3–16 (cit. on p. 3).

[28] A. Ehrenfeucht et al. *Qualitative and Quantitative Aspects of a Model for Processes Inspired by the Functioning of a Living Cell*. Wiley Online Library, 2011 (cit. on p. 3).

[29] A. Ehrenfeucht and G. Rozenberg. "Introducing time in reaction systems". In: *Theoretical Computer Science* 410.4-5 (2009), pp. 310–322 (cit. on pp. 3, 4, 13).

[30] R. Brijder et al. "A tour of reaction systems". In: *International Journal of Foundations of Computer Science* 22.07 (2011), pp. 1499–1517 (cit. on pp. 3, 71).

[31] A. Ehrenfeucht, M. Main, and G. Rozenberg. "Combinatorics of life and death for reaction systems". In: *International Journal of Foundations of Computer Science* 21.03 (2010), pp. 345–356 (cit. on p. 4).

[32] R. Brijder, A. Ehrenfeucht, and G. Rozenberg. "Reaction systems with duration". In: *Computation, cooperation, and life*. Springer, 2011, pp. 191–202 (cit. on pp. 4, 62).

[33] A. Ehrenfeucht, I. Petre, and G. Rozenberg. "Reaction systems: a model of computation inspired by the functioning of the living cell". In: *THE ROLE OF THEORY IN COMPUTER SCIENCE: Essays Dedicated to Janusz Brzozowski*. World Scientific, 2017, pp. 1–32 (cit. on p. 4).

[34] A. Salomaa. "Minimal reaction systems: Duration and blips". In: *Theoretical Computer Science* 682 (2017), pp. 208–216 (cit. on pp. 4, 62).

[35] S. Azimi, B. Iancu, and I. Petre. "Reaction system models for the heat shock response". In: *Fundamenta Informaticae* 131.3-4 (2014), pp. 299–312 (cit. on pp. 4, 14, 67).

[36] S. Azimi et al. "Reaction systems models for the self-assembly of intermediate filaments". In: *Annals of University of Bucharest* LXII (2015), pp. 9–24 (cit. on pp. 4, 58, 62, 63).

[37] S. Azimi et al. "Multi-stability, limit cycles, and period-doubling bifurcation with reaction systems". In: *International Journal of Foundations of Computer Science* 28.08 (2017), pp. 1007–1020 (cit. on p. 4).

[38] A. Salomaa. "Applications of the Chinese remainder theorem to reaction systems with duration". In: *Theoretical Computer Science* 598 (2015), pp. 15–22 (cit. on p. 4).

[39] A. Ehrenfeucht, M. Main, and G. Rozenberg. "Functions defined by reaction systems". In: *International Journal of Foundations of Computer Science* 22.01 (2011), pp. 167–178 (cit. on p. 4).

[40] A. Salomaa. "Functions and sequences generated by reaction systems". In: *Theoretical Computer Science* 466 (2012), pp. 87–96 (cit. on p. 4).

[41] E. Formenti, L. Manzoni, and A. E. Porreca. "Fixed points and attractors of reaction systems". In: *Conference on Computability in Europe*. Springer. 2014, pp. 194–203 (cit. on p. 4).

[42] G. M. Cooper and R. E. Hausman. *The cell: A Molecular approach*. ASM Press and Sinauer Sunderland Associates, Inc., 2007 (cit. on p. 4).

[43] J. L. Goldstein, R. G. W. Anderson, and M. S. Brown. "Coated pits, coated vesicles, and receptor-mediated endocytosis". In: *Nature* 279 (5715 1979), pp. 679–685 (cit. on p. 4).

[44] G. Puaun, G. Rozenberg, and A. Salomaa. *DNA Computing - New Computing Paradigms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 1998. ISBN: 978-3-540-64196-4. DOI: 10.1007/978-3-662-03563-4 (cit. on p. 4).

[45] G. Puaun. "Computing with Membranes: An Introduction". In: *Bulletin of the EATCS* 67 (1999), pp. 139–152 (cit. on pp. 4, 15).

[46] G. Păun. *Membrane Computing: An Introduction*. Springer, 2002 (cit. on pp. 4, 15, 39, 40).

[47] G. Puaun. "Computing with Membranes (P Systems): A Variant". In: *Int. J. Found. Comput. Sci.* 11.1 (2000), pp. 167–181. DOI: 10.1142/S0129054100000090 (cit. on p. 4).

[48] P. Dittrich, J. Ziegler, and W. Banzhaf. "Artificial chemistries—a review". In: *Artificial life* 7 (3 2001), pp. 225–275 (cit. on p. 4).

[49] S. N. Krishna and R. Rama. "P Systems with Replicated Rewriting". In: *Journal of Automata, Languages and Combinatorics* 6.3 (2001), pp. 345–350. DOI: 10.25596/jalc-2001-345 (cit. on p. 5).

[50] C. Martín-Vide et al. "Tissue P systems". In: *Theoretical Computer Science* 296.2 (2003), pp. 295–326. DOI: 10.1016/S0304-3975(02)00659-X (cit. on p. 5).

[51] F. Bernardini and M. Gheorghe. "Population P Systems". In: *Journal of Universal Computer Science* 10.5 (2004), pp. 509–539. DOI: 10.3217/jucs-010-05-0509 (cit. on p. 5).

[52] R. Freund, G. Păun, and M. J. Pérez-Jiménez. "Tissue P systems with channel states". In: *Theoretical Computer Science* 330.1 (2005), pp. 101–116. DOI: 10.1016/j.tcs.2004.09.013 (cit. on p. 5).

[53] M. Ionescu et al. "Computing with Spiking Neural P Systems: Traces and Small Universal Systems". In: *DNA Computing, 12th International Meeting on DNA Computing, DNA12, Seoul, Korea, June 5-9, 2006*. Vol. 4287. Lecture Notes in Computer Science. Springer, 2006, pp. 1–16. DOI: 10.1007/11925903_1. URL: https://doi.org/10.1007/11925903_1 (cit. on p. 5).

[54] M. García-Arnau et al. "Spiking Neural P Systems: Stronger Normal Forms". In: *International Journal of Unconventional Computing* 5.5 (2009), pp. 411–425. URL: http://www.oldcitypublishing.com/journals/ijuc-home/ijuc-issue-contents/ijuc-volume-5-number-5-2009/ijuc-5-5-p-411-425/ (cit. on p. 5).

[55] Y. Jiang, Y. Su, and F. Luo. "An improved universal spiking neural P system with generalized use of rules". In: *Journal of Membrane Computing* 1.4 (2019), pp. 270–278. ISSN: 2523-8914. DOI: 10.1007/s41965-019-00025-y. URL: https://doi.org/10.1007/s41965-019-00025-y (cit. on p. 5).

[56] Y. Suzuki et al. "Artificial life applications of a class of P systems: Abstract rewriting systems on multisets". In: *Workshop on Membrane Computing*. Springer. 2000, pp. 299–346 (cit. on pp. 5, 101).

[57] T. Y. Nishida. "A membrane computing model of photosynthesis". In: Springer, 2006, pp. 181–202 (cit. on pp. 5, 101).

[58] M. Cavaliere and I. I. Ardelean. "Modeling Respiration in Bacteria and Respiration/PhotosynthesisInteraction in Cyanobacteria Using a P System Simulator". In: *Applications of Membrane Computing*. 2006, pp. 129–158 (cit. on p. 5).

[59] Y. Suzuki and H. Tanaka. "Modeling p53 Signaling Pathways by Using Multiset Processing". In: *Applications of Membrane Computing*. Ed. by G. Ciobanu, M. J. Pérez-Jiménez, and G. Paun. Natural Computing Series. Springer, 2006, pp. 203–214. DOI: 10.1007/3-540-29937-8_7. URL: https://doi.org/10.1007/3-540-29937-8_7 (cit. on pp. 5, 101).

[60] G. Ciobanu. "Modeling Cell-Mediated Immunity by Means of P Systems". In: *Applications of Membrane Computing*. 2006, pp. 159–180 (cit. on p. 5).

[61] M. A. Martínez-del-Amor et al. "Parallel simulation of Population Dynamics P systems: updates and roadmap". In: *Natural Computing* 15.4 (2016), pp. 565–573. DOI: 10.1007/s11047-016-9566-1 (cit. on pp. 5, 101).

[62] G. Ciobanu, G. Păun, and M. J. Pérez-Jiménez. *Applications of membrane computing*. Vol. 17. Springer, 2006 (cit. on p. 5).

[63] K. Compton and S. Hauck. "Reconfigurable Computing: A Survey of Systems and Software". In: *ACM Computing Surveys* 34.2 (2002), 171–210. ISSN: 0360-0300. DOI: 10.1145/508352.508353. URL: https://doi.org/10.1145/508352.508353 (cit. on p. 5).

[64] G. Estrin et al. "Parallel Processing in a Restructurable Computer System". In: *IEEE Transactions on Electronic Computers* EC-12.6 (1964), pp. 747–755 (cit. on p. 5).

[65] Wikipedia. *Reconfigurable Computing*. 2015. URL: https://en.wikipedia.org/wiki/Reconfigurable_computing (cit. on p. 5).

[66] S. Hauck and A. DeHon. *Reconfigurable Computing-The theory and practice of FPGA-based computation*. Elsevier Morgan Kaufmann, 2008 (cit. on pp. 5, 31).

[67] J. R. Hauser and J. Wawrzynek. "Garp: a MIPS processor with a reconfigurable coprocessor". In: *5th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '97), 16-18 April 1997, Napa Valley, CA, USA*. IEEE Computer Society, 1997, pp. 12–21. DOI: 10.1109/FPGA.1997.624600 (cit. on p. 5).

[68] T. J. Callahan, J. R. Hauser, and J. Wawrzynek. "The Garp architecture and C compiler". In: *Computer* 33.4 (2000), pp. 62–69 (cit. on p. 5).

[69] S. C. Goldstein et al. "PipeRench: a coprocessor for streaming multimedia acceleration". In: *Proceedings of the 26th International Symposium on Computer Architecture (Cat. No.99CB36367)*. 1999, pp. 28–39 (cit. on p. 5).

[70] Y. C. Chou et al. "PipeRench implementation of the instruction path coprocessor". In: *Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 33, Monterey, California, USA, December 10-13, 2000*. Ed. by A. Wolfe and M. S. Schlansker. ACM/IEEE Computer Society, 2000, pp. 147–158. DOI: 10.1109/MICRO.2000.898066 (cit. on p. 5).

[71] C. Ebeling, D. C. Cronquist, and P. Franklin. "RaPiD - Reconfigurable Pipelined Datapath". In: *Field-Programmable Logic, Smart Applications, New Paradigms and Compilers, 6th International Workshop on Field-Programmable Logic, FPL '96, Darmstadt, Germany, September 23-25, 1996, Proceedings*. Ed. by R. W. Hartenstein and M. Glesner. Vol. 1142. Lecture Notes in Computer Science. Springer, 1996, pp. 126–135. DOI: 10.1007/3-540-61730-2_13. URL: https://doi.org/10.1007/3-540-61730-2_13 (cit. on p. 6).

[72] H. Schmit. "Incremental reconfiguration for pipelined applications". In: *5th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '97), 16-18 April 1997, Napa Valley, CA, USA*. IEEE Computer Society, 1997, pp. 47–55. DOI: 10.1109/FPGA.1997.624604 (cit. on p. 6).

[73] D. C. Cronquist et al. "Architecture Design of Reconfigurable Pipelined Datapaths". In: *18th Conference on Advanced Research in VLSI (ARVLSI '99), 21-24 March 1999, Atlanta, GA, USA*. IEEE Computer Society, 1999, pp. 23–41. DOI: 10.1109/ARVLSI.1999.756035 (cit. on p. 6).

[74] D. Wilson. "Chameleon takes on FPGAs, ASICs". In: *Electronic Business Asia, EDNOnline Magazine* (2000). URL: http://www.edn.com/article/CA50551.html?partner=enews (cit. on p. 6).

[75] Z. A. Ye et al. "CHIMAERA: a high-performance architecture with a tightly-coupled reconfigurable functional unit". In: *27th International Symposium on Computer Architecture (ISCA 2000), June 10-14, 2000, Vancouver, BC, Canada*. Ed. by A. D. Berenbaum and J. S. Emer. IEEE Computer Society, 2000, pp. 225–235. DOI: 10.1109/ISCA.2000.854393 (cit. on p. 6).

[76] J. E. Carrillo and P. Chow. "The effect of reconfigurable units in superscalar processors". In: *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA 2001, Monterey, CA, USA, February 11-13, 2001*. Ed. by S. Hauck, M. D. F. Schlag, and R. Tessier. ACM, 2001, pp. 141–150. DOI: 10.1145/360276.360328 (cit. on p. 6).

[77] S. Hauck et al. "The Chimaera reconfigurable functional unit". In: *IEEE Trans. Very Large Scale Integr. Syst.* 12.2 (2004), pp. 206–217 (cit. on p. 6).

[78] G. Estrin. "Organization of Computer Systems: The Fixed plus Variable Structure Computer". In: *Papers Presented at the May 3-5, 1960, Western Joint IRE-AIEE-ACM Computer Conference*. 1960, 33–40. ISBN: 9781450378697. DOI: 10.1145/1460361.1460365. URL: https://doi.org/10.1145/1460361.1460365 (cit. on p. 6).

[79] G. Estrin et al. "Parallel Processing in a Restructurable Computer System". In: *IEEE Trans. Electronic Computers* 12.6 (1963), pp. 747–755. DOI: 10.1109/PGEC.1963.263558 (cit. on p. 6).

[80] G. Estrin. "Reconfigurable Computer Origins: The UCLA Fixed-Plus-Variable (F+V) Structure Computer". In: *IEEE Annals of the History of Computing* 24.4 (2002), pp. 3–9. DOI: 10.1109/MAHC.2002.1114865 (cit. on p. 6).

[81] P. Bertin, D. Roncin, and J. Vuillemin. *Introduction to programmable active memories*. Tech. rep. DEC Paris Research Laboratory, 1989 (cit. on p. 6).

[82] S. Casselman. "Virtual computing and the virtual computer". In: *IEEE Workshop on FPGAs for Custom Computing Machines*. May 1993, pp. 43–48. DOI: 10.1109/FPGA.1993.279480 (cit. on p. 6).

[83] M. B. Gokhale et al. "SPLASH: A Reconfigurable Linear Logic Array". In: *Proceedings of the 1990 International Conference on Parallel Processing, Urbana-Champaign, IL, USA, August 1990. Volume 1: Architecture*. Ed. by B. W. Wah. Pennsylvania State University Press, 1990, pp. 526–532 (cit. on p. 6).

[84] P. M. Athanas and H. F. Silverman. "Processor Reconfiguration Through Instruction-Set Metamorphosis". In: *IEEE Computer* 26.3 (1993), pp. 11–18. DOI: 10.1109/2.204677 (cit. on p. 6).

[85] T. A. Kean. "Configurable Logic: A Dynamically Programmable Cellular Architectureand Its VLSI Implementation". PhD thesis. University of Edinburgh, 1989. URL: https://era.ed.ac.uk/handle/1842/271 (cit. on p. 6).

[86] P. Bottoni et al. "Membrane systems with promoters/inhibitors". In: *Acta Inf.* 38.10 (2002), pp. 695–720. DOI: 10.1007/s00236-002-0090-7 (cit. on p. 16).

[87] O. Agrigoroaiei and G. Ciobanu. "Rewriting Logic Specification of Membrane Systems with Promoters and Inhibitors". In: *Proceedings of the Seventh International Workshop on Rewriting Logic and its Applications, WRLA 2008, Budapest, Hungary, March 29-30, 2008*. Ed. by G. Rosu. Vol. 238. Electronic Notes in Theoretical Computer Science 3. Elsevier, 2008, pp. 5–22. DOI: 10.1016/j.entcs.2009.05.010 (cit. on p. 16).

[88] J. Kleijn and M. Koutny. "Processes of membrane systems with promoters and inhibitors". In: *Theor. Comput. Sci.* 404.1-2 (2008), pp. 112–126. DOI: 10.1016/j.tcs.2008.04.006 (cit. on p. 16).

[89] V. T. T. Nguyen. "An Implementation of the Parallelism,Distribution and Nondeterminism of Membrane Computing Models on Reconfigurable Hardware". PhD thesis. University of South Australia, 2010 (cit. on pp. 16, 44, 45).

[90] S. Verlan and J. Quiros. "Fast Hardware Implementations of P Systems". In: *Membrane Computing - 13th International Conference, CMC 2012, Budapest, Hungary, August 28-31, 2012, Revised Selected Papers*. Ed. by E. Csuhaj-Varjú et al. Vol. 7762. Lecture Notes in Computer Science. Springer, 2012, pp. 404–423. DOI: 10.1007/978-3-642-36751-9_27 (cit. on pp. 16, 26, 40–42, 48, 49).

[91] R. Freund and S. Verlan. "A Formal Framework for Static (Tissue) P Systems". In: *Membrane Computing, 8th International Workshop, WMC 2007, Thessaloniki, Greece, June 25-28, 2007 Revised Selected and Invited Papers*. Ed. by G. Eleftherakis et al. Vol. 4860. Lecture Notes in Computer Science. Springer, 2007, pp. 271–284. DOI: 10.1007/978-3-540-77312-2_17 (cit. on pp. 17, 19, 21, 23, 40, 80, 88).

[92] S. Verlan. *Study of Language-Theoretic Computational Paradigms Inspired by Biology*. Tech. rep. Université Paris Est, 2010 (cit. on pp. 17, 19).

[93] S. Verlan. "Using the Formal Framework for P Systems". In: *Membrane Computing - 14th International Conference, CMC 2013, Chişinău, Republic of Moldova, August 20-23, 2013, Revised Selected Papers*. Ed. by A. Alhazov et al. Vol. 8340. Lecture Notes in Computer Science. Springer, 2013, pp. 56–79. DOI: 10.1007/978-3-642-54239-8_6 (cit. on pp. 17, 19).

[94] R. Freund et al. "Flattening in (Tissue) P Systems". In: *Membrane Computing - 14th International Conference, CMC 2013, Chişinău, Republic of Moldova, August 20-23, 2013, Revised Selected Papers*. Ed. by A. Alhazov et al. Vol. 8340. Lecture Notes in Computer Science. Springer, 2013, pp. 173–188. DOI: 10.1007/978-3-642-54239-8_13 (cit. on pp. 17, 21, 40).

[95] R. Freund et al. "A Formalization of Membrane Systems with Dynamically Evolving Structures". In: *International Journal of Computer Mathematics* 90.4 (2013), pp. 801–815. DOI: 10.1080/00207160.2012.748899. eprint: http://www.tandfonline.com/doi/pdf/10.1080/00207160.2012.748899. URL: http://www.tandfonline.com/doi/abs/10.1080/00207160.2012.748899 (cit. on p. 19).

[96] I. Kaliszewski, J. Miroforidis, and D. Podkopaev. *Multiple Criteria Decision Making by Multiobjective Optimization*. Springer, 2016 (cit. on p. 20).

[97] A. Alhazov. "Maximally Parallel Multiset-Rewriting Systems: Browsing the Configurations". In: *Proceedings of the Third Brainstorming Week on Membrane Computing*. RGNC Report 01/2005. 2005, pp. 1–10 (cit. on p. 21).

[98] M. Ehrgott. *Multicriteria Optimization*. Springer, 2005. ISBN: 978-3-540-21398-7. DOI: 10.1007/3-540-27659-9. URL: https://doi.org/10.1007/3-540-27659-9 (cit. on p. 22).

[99] O. Agrigoroaiei, G. Ciobanu, and A. Resios. "Evolving by Maximizing the Number of Rules: Complexity Study". In: *Membrane Computing: 10th International Workshop, WMC 2009, Curtea de Arges, Romania, August 24-27, 2009. Revised Selected and Invited Papers*. Ed. by G. Păun et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 149–157. ISBN: 978-3-642-11467-0. DOI: 10.1007/978-3-642-11467-0_11. URL: https://doi.org/10.1007/978-3-642-11467-0_11 (cit. on p. 22).

[100] R. Reina-Molina, D. Díaz-Pernil, and M. Gutiérrez-Naranjo. "Integer Linear Programming for Tissue-like P Systems". In: *Proceedings of the Ninth Brainstorming Week on Membrane Computing*. 2011 (cit. on p. 22).

[101] G. Ciobanu and A. Resios. "Complexity of evolution in maximum cooperative P systems". In: *Natural Computing* 8.4 (2009), p. 807. ISSN: 1572-9796. DOI: 10.1007/s11047-009-9107-2. URL: https://doi.org/10.1007/s11047-009-9107-2 (cit. on pp. 23, 39).

[102] G. Ciobanu, S. Marcus, and G. Păun. "New Strategies of Using the Rules of a P System in a Maximal Way: Power and Complexity". In: *Romanian Journal of Information Science and Technology* 12.2 (2009), pp. 157–173 (cit. on p. 23).

[103] V. Nguyen, D. Kearney, and G. Gioiosa. "An Algorithm for Non-deterministic Object Distribution in P Systems and Its Implementation in Hardware". In: *Membrane Computing - 9th International Workshop, WMC 2008, Edinburgh, UK, July 28-31, 2008, Revised Selected and Invited Papers*. Ed. by D. W. Corne et al. Vol. 5391. Lecture Notes in Computer Science. Springer, 2008, pp. 325–354. DOI: 10.1007/978-3-540-95885-7_24 (cit. on pp. 23, 26, 41).

[104] A. Arteta, L. Fernandez, and J. Gil. "Algorithm for Application of Evolution Rules Based on Linear Diofantic Equations". In: *SYNASC 2008, 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania, 26-29 September 2008*. Ed. by V. Negru et al. IEEE Computer Society, 2008, pp. 496–500. DOI: 10.1109/SYNASC.2008.31 (cit. on p. 23).

[105] V. J. Martínez et al. "Hardware Implementation of a Bounded Algorithm for Application of Rules in a Transition P-System". In: *8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2006), 26-29 September 2006, Timisoara, Romania*. Ed. by V. Negru et al. IEEE Computer Society, 2006, pp. 343–349. DOI: 10.1109/SYNASC.2006.42 (cit. on pp. 26, 40).

[106] J. A. Tejedor et al. "Optimizing Evolution Rules Application and Communication Times in Membrane Systems Implementation". In: *Membrane Computing, 8th International Workshop, WMC 2007, Thessaloniki, Greece, June 25-28, 2007 Revised Selected and Invited Papers*. Ed. by G. Eleftherakis et al. Vol. 4860. Lecture Notes in Computer Science. Springer, 2007, pp. 298–319. DOI: 10.1007/978-3-540-77312-2_19 (cit. on p. 26).

[107] L. Fernandez et al. "Decision Trees for Applicability of Evolution Rules in Transition P Systems". In: *Information Theories and Applications* 14.3 (2007), pp. 223–230. URL: http://hdl.handle.net/10525/684 (cit. on pp. 26, 41).

[108] D. Kirk and W.-M. Hwu. *Programming Massively Parallel Processors: A Hands On Approach*. USA: Morgan Kaufmann, 2010. ISBN: 978-0-12-381472-2 (cit. on p. 28).

[109] N. Wilt. *The CUDA Handbook:A Comprehensive Guide to GPU Programming*. Addison Wesley, 2013 (cit. on pp. 28, 36).

[110] NVIDIA. *CUDA C++ Programming Guide*. 2020. URL: https://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf (cit. on p. 29).

[111] M. A. Martínez-del-Amor et al. "Simulating P Systems on GPU Devices: A Survey". In: *Fundam. Inform.* 136.3 (2015), pp. 269–284. DOI: 10.3233/FI-2015-1157 (cit. on p. 28).

[112] NVIDIA. *CUDA Toolkit Documentation*. 2019. URL: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html (cit. on p. 29).

[113] C. M. Maxfield. *The Design Warrior's Guide to FPGA*. Elsevier, 2004 (cit. on pp. 30, 34).

[114] C. M. Maxfield. *FPGAs World Class Designs*. Ed. by C. Maxfield. Elsevier, 2009 (cit. on p. 30).

[115] A. Pavlov and M. Sachdev. *CMOS SRAM Circuit Designand Parametric Testin Nano-Scaled Technologies*. Springer, 2008 (cit. on p. 30).

[116] V. Singh. "Equivalent forms of single-operational transconductance amplifier RC oscillators with application to grounded-capacitor oscillators". In: *IET Circuits, Devices & Systems* 4.2 (2010), pp. 123–130. DOI: 10.1049/iet-cds.2009.0146 (cit. on p. 31).

[117] A. Paidimarri et al. "An RC Oscillator With Comparator Offset Cancellation". In: *J. Solid-State Circuits* 51.8 (2016), pp. 1866–1877. DOI: 10.1109/JSSC.2016.2559508 (cit. on p. 31).

[118] L. A. C. Ahumada et al. "Evaluation of Hyaluronic Acid Dilutions at Different Concentrations Using a Quartz Crystal Resonator (QCR) for the Potential Diagnosis of Arthritic Diseases". In: *Sensors* 16.11 (2016), p. 1959. DOI: 10.3390/s16111959 (cit. on p. 31).

[119] Y. Murozaki, S. Sakuma, and F. Arai. "Improvement of the Measurement Range and Temperature Characteristics of a Load Sensor Using a Quartz Crystal Resonator with All Crystal Layer Components". In: *Sensors* 17.5 (2017), p. 1067. DOI: 10.3390/s17051067 (cit. on p. 31).

[120] W. Brinkman, D. Haggan, and W. Troutman. "A history of the invention of the transistor and where it will lead us". In: *IEEE Journal of Solid-State Circuits* 32 (12 1997), pp. 1858–1865. ISSN: 1558-173X. DOI: 10.1109/4.643644 (cit. on p. 34).

[121] M. J. M. V. P. Heuring. *Computer Architecture and Organization*. John Wiley and Sons Inc, 2007. URL: https://www.niecdelhi.ac.in/uploads/Notes/btech/5sem/cse/CA_Notes.pdf (cit. on p. 34).

[122] K. Keutzer and S. Malik. "Register Transfer Level Synthesis: From Theory to Practice". In: *9th International Conference on VLSI Design (VLSI Design 1996), 3-6 January 1996, Bangalore, India*. IEEE Computer Society, 1996, p. 2. DOI: 10.1109/VLSID.1996.10009 (cit. on p. 34).

[123] J. Bhasker. "Synthesis at the Register Transfer Level and the Behavioral Level". In: *The VLSI Handbook*. Ed. by W. Chen. CRC Press, 1999. DOI: 10.1201/9781420049671.ch75 (cit. on p. 34).

[124] O. Port and Y. Etsion. "Hardware Description Beyond Register-Transfer Level Languages". In: *FPGA '20: The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Seaside, CA, USA, February 23-25, 2020*. Ed. by S. Neuendorffer and L. Shannon. ACM, 2020, p. 312. DOI: 10.1145/3373087.3375377 (cit. on p. 34).

[125] Xilinx, ed. *Vivado Design Suite User Guide: Using the Vivado IDE*. 2019. URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug893-vivado-ide.pdf (cit. on p. 34).

[126] Xilinx, ed. *Vivado Design Suite User Guide: Design Flows Overview*. 2019. URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug892-vivado-design-flows-overview.pdf (cit. on p. 34).

[127] Xilinx, ed. *Vivado Design Suite User Guide: System-Level Design Entry*. 2019. URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug895-vivado-system-level-design-entry.pdf (cit. on p. 34).

[128] J. Bhasker. *A Verilog HDL Primer*. Ed. by S. G. Press. Second. 1058 Treeline Drive, Allentawn, PA 18103: Star Galaxy Publishing, 1999. ISBN: 0-9650391-7-X (cit. on p. 34).

[129] Xilinx, ed. *Vivado Design Suite User Guide: Logic Simulation*. 2019. URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug900-vivado-logic-simulation.pdf (cit. on p. 34).

[130] Xilinx, ed. *Vivado Design Suite Tutorial: Logic Simulation*. 2020. URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_1/ug937-vivado-design-suite-simulation-tutorial.pdf (cit. on p. 34).

[131] Xlinx, ed. *Vivado Design Suite User Guide: Synthesis*. 2019. URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_1/ug901-vivado-synthesis.pdf (cit. on p. 35).

[132] *Vivado Design Suite User Guide: Using Constraints*. 2019. URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug903-vivado-using-constraints.pdf (cit. on pp. 35, 95).

[133] Xilinx, ed. *Vivado Design Suite Tutorial: Using Constraints*. 2019. URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug945-vivado-using-constraints-tutorial.pdf (cit. on p. 35).

[134] Xilinx, ed. *Vivado Design Suite User Guide: Implementation*. 2019. URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_1/ug904-vivado-implementation.pdf (cit. on p. 35).

[135] Xilinx. *Vivado Design Suite*. 2019. URL: https://www.xilinx.com/products/design-tools/vivado.html#documentation (cit. on p. 35).

[136] Xilinx, ed. *Vivado Design Suite User Guide: Programming and Debugging*. 2019. URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug908-vivado-programming-debugging.pdf (cit. on p. 35).

[137] Xilinx, ed. *Vivado Design Suite Tutorial: Programming and Debugging*. 2019. URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug936-vivado-tutorial-programming-debugging.pdf (cit. on p. 35).

[138] J. Sanders and E. Kandrot. *CUDA by Example: An IntroductIon to General-Purpose GPU ProgrammIng*. Addison Wesley, 2010 (cit. on p. 36).

[139] S. Azimi et al. "Dependency graphs and mass conservation in reaction systems". In: *Theor. Comput. Sci.* 598 (2015), pp. 23–39. DOI: 10.1016/j.tcs.2015.02.014 (cit. on p. 36).

[140] V. Rogojin, S. Azimi, and I. Petre. *Web interface for Basic Reaction System Simulator*. 2015. URL: http://combio.abo.fi/research/reaction-systems/reaction-system-simulator/ (cit. on p. 36).

[141] S. Ivanov et al. "WEBRSIM: A Web-Based Reaction Systems Simulator". In: *Enjoying Natural Computing - Essays Dedicated to Mario de Jesús Pérez-Jiménez on the Occasion of His 70th Birthday*. Ed. by C. G. Díaz et al. Vol. 11270. Lecture Notes in Computer Science. Springer, 2018, pp. 170–181. DOI: 10.1007/978-3-030-00265-7_14. URL: https://doi.org/10.1007/978-3-030-00265-7_14 (cit. on p. 36).

[142] S. Azimi et al. "Complexity of model checking for reaction systems". In: *Theor. Comput. Sci.* 623 (2016), pp. 103–113. DOI: 10.1016/j.tcs.2015.11.040 (cit. on p. 37).

[143] S. Ivanov et al. *GitHub – scolobb/brsim: A Basic Reaction System Simulator*. 2015. URL: https://github.com/scolobb/brsim (cit. on p. 37).

[144] A. Meski, W. Penczek, and G. Rozenberg. "Model checking temporal properties of reaction systems". In: *Inf. Sci.* 313 (2015), pp. 22–42. DOI: 10.1016/j.ins.2015.03.048 (cit. on p. 37).

[145] M. S. Nobile et al. "Efficient Simulation of Reaction Systems on Graphics Processing Units". In: *Fundam. Inform.* 154.1-4 (2017), pp. 307–321. DOI: 10.3233/FI-2017-1568 (cit. on pp. 37, 55, 69).

[146] M. S. Nobile, A. E. Porreca, and S. Spolaor. *HERESY GitHub Repository*. 2017. URL: https://github.com/aresio/HERESY/ (cit. on p. 37).

[147] L. Corolli et al. "An excursion in reaction systems: From computer science to biology". In: *Theor. Comput. Sci.* 454 (2012), pp. 95–108. DOI: 10.1016/j.tcs.2012.04.003 (cit. on p. 37).

[148] T. Helikar et al. "A comprehensive, multi-scale dynamical model of ErbB receptor signal transduction in human mammary epithelial cells". In: *PLoS ONE* 8.4 (2013), e61757 (cit. on pp. 37, 69).

[149] J. K. Li, J. Chou, and K. Ma. "High performance heterogeneous computing for collaborative visual analysis". In: *SIGGRAPH Asia 2015 Visualization in High Performance Computing, Kobe, Japan, November 2-6, 2015*. ACM, 2015, 12:1–12:4. DOI: 10.1145/2818517.2818534 (cit. on p. 38).

[150] T. Katagiri. "High-Performance Computing Basics". In: *The Art of High Performance Computing for Computational Science, Vol. 1, Techniques of Speedup and Parallelization for General Purposes*. Ed. by M. Geshi. Springer, 2019, pp. 1–25. DOI: 10.1007/978-981-13-6194-4_1. URL: https://doi.org/10.1007/978-981-13-6194-4_1 (cit. on p. 38).

[151] P. K. Bondyopadhyay. "Moore's law governs the silicon revolution". In: *Proceedings of the IEEE* 86 (1 1998), pp. 78–81. ISSN: 1558-2256. DOI: 10.1109/5.658761 (cit. on p. 38).

[152] E. P. DeBenedictis. "It's Time to Redefine Moore's Law Again". In: *Computer* 50 (2 2017), pp. 72–75. ISSN: 1558-0814. DOI: 10.1109/MC.2017.34 (cit. on p. 38).

[153] N. Mathur. "Beyond the silicon roadmap". In: *Nature* 419 (Oct. 2002), pp. 573–575 (cit. on p. 38).

[154] P. Wesling. "The Heterogeneous Integration Roadmap: Enabling Technology for Systems of the Future". In: HI, USA. HI, USA: IEEE, 2020, pp. 1–4. ISBN: 978-1-7281-4584-6. DOI: 10.23919/PanPacific48324.2020.9059347 (cit. on p. 38).

[155] P. Z. Roman Trobec Marián Vajteršic. *Parallel Computing*. Ed. by P. Zinterhof. Springer, 2009. DOI: 10.1007/978-1-84882-409-6_1 (cit. on p. 38).

[156] H. K. Anil Sethi. "Multicore Processor Technology-Advantages and Chanllenges". In: *International Journal of Research in Engineering and Technology* 4.9 (Sept. 2015), pp. 87–89. ISSN: 2319-1163 (cit. on p. 38).

[157] G. Zhang, M. J. Préz-Jimnéz, and M. Gheorghe. *Real-life Applications with Membrane Computing*. Springer, 2017. ISBN: 3319559877, 9783319559872 (cit. on pp. 38, 101).

[158] R. U. F. Varela H. Maturana. "Autopoiesis: The Organization of Living Systems, its Characterization and a Model". In: *BioSystems* 5 (1974), pp. 187–196 (cit. on p. 38).

[159] G. Ciobanu and G. Wenyuan. "P systems running on a cluster of computers". In: *International Workshop on Membrane Computing*. Springer. 2003, pp. 123–139 (cit. on p. 39).

[160] B. Petreska and C. Teuscher. "A Reconfigurable Hardware Membrane System". In: *Membrane Computing, International Workshop, WMC 2003, Tarragona, Spain, July 17-22, 2003, Revised Papers*. Ed. by C. Martín-Vide et al. Vol. 2933. Lecture Notes in Computer Science. Springer, 2003, pp. 269–285. DOI: 10.1007/978-3-540-24619-0_20 (cit. on pp. 40, 42, 43).

[161] V. Nguyen, D. Kearney, and G. Gioiosa. "A Region-Oriented Hardware Implementation for Membrane Computing Applications". In: *Membrane Computing, 10th International Workshop, WMC 2009, Curtea de Arges, Romania, August 24-27, 2009. Revised Selected and Invited Papers*. Ed. by G. Paun et al. Vol. 5957. Lecture Notes in Computer Science. Springer, 2009, pp. 385–409. DOI: 10.1007/978-3-642-11467-0_27 (cit. on pp. 40, 41).

[162] V. Nguyen, D. Kearney, and G. Gioiosa. "Balancing Performance, Flexibility, and Scalability in a Parallel Computing Platform for Membrane Computing Applications". In: *Membrane Computing, 8th International Workshop, WMC 2007, Thessaloniki, Greece, June 25-28, 2007 Revised Selected and Invited Papers*. Ed. by G. Eleftherakis et al. Vol. 4860. Lecture Notes in Computer Science. Springer, 2007, pp. 385–413. DOI: 10.1007/978-3-540-77312-2_24 (cit. on pp. 40, 45).

[163] V. Nguyen, D. Kearney, and G. Gioiosa. "An Implementation of Membrane Computing Using Reconfigurable Hardware". In: *Computing and Informatics* 27.3+ (2008), pp. 551–569. URL: http://www.sav.sk/index.php?lang=en&charset=ascii&doc=journal&part=list_articles&journal_issue_no=11112123#abstract_4951 (cit. on pp. 40, 45).

[164] J. Quiros et al. "Fast Hardware Implementations of Static P Systems". In: *Computing and Informatics* 35.3 (2016), pp. 687–718. URL: http://www.cai.sk/ojs/index.php/cai/article/view/1665 (cit. on pp. 40–42, 48).

[165] G. Bravo et al. "A Hierarchical Architecture with Parallel Comunication for Implementing P Systems". In: *Information Technologies and Knowledge* 2.1 (2008), pp. 43–48. URL: http://hdl.handle.net/10525/269 (cit. on p. 40).

[166] A. Gutierrez et al. "Hardware and Software Architecture for Implementing Membrane Systems: A Case of Study to Transition P Systems". In: *DNA Computing, 13th International Meeting on DNA Computing, DNA13, Memphis, TN, USA, June 4-8, 2007, Revised Selected Papers*. Ed. by M. H. Garzon and H. Yan. Vol. 4848. Lecture Notes in Computer Science. Springer, 2007, pp. 211–220. DOI: 10.1007/978-3-540-77962-9_22 (cit. on p. 40).

[167] V. Martínez, S. Alonso, and A. Gutiérrez. "Hardware Circuit for the Application of Evolution Rules in a Transition P-system". In: *Artificial Life and Robotics* 15.1 (2010), pp. 89–92. ISSN: 1614-7456. DOI: 10.1007/s10015-010-0774-y. URL: https://doi.org/10.1007/s10015-010-0774-y (cit. on p. 40).

[168] A. V. Baranda et al. "Towards an Electronic Implementation of Membrane Computing: A Formal Description of Non-deterministic Evolution in Transition P Systems". In: *DNA Computing, 7th International Workshop on DNA-Based Computers, DNA7, Tampa, Florida, USA, June 10-13, 2001, Revised Papers*. Ed. by N. Jonoska and N. C. Seeman. Vol. 2340. Lecture Notes in Computer Science. Springer, 2001, pp. 350–359. DOI: 10.1007/3-540-48017-X_33 (cit. on p. 40).

[169] L. Fernando et al. "Massively Parallel Algorithm for Evolution Rules Application in Transition P Systems". In: *Proceedings of 7th Workshop on Membrane Computing*. Ed. by H. J. Hoogeboom, G. Păun, and G. Rozenberg. Lorentz Center, Universiteit Leiden, 2006, pp. 337–343 (cit. on p. 41).

[170] J. A. Tejedor et al. "Algorithm of Active Rule Elimination for Application of Evolution Rules". In: *Proceedings of the 8th Conference on 8th WSEAS International Conference on Evolutionary Computing*. Vol. 8. 2007, pp. 259–267 (cit. on p. 41).

[171] J. A. Tejedor et al. "Algorithm of rules applications based on competitiveness of evolution rules". In: *Proceedings of the 8th Workshop on Membrane Computing*. 2007, pp. 567–580 (cit. on p. 41).

[172] F. J. G. Rubio et al. "Delimited Massively Parallel Algorithm based on Rules Elimination for Application of Active Rules in Transition P Systems". In: *Proceedings of the Fifth International Conference ?Information Research and Applications? i.TECH 2007*. Ed. by K. Markov and K. Ivanova. Vol. Vol. 1. Bulgaria: Institute of Information Theories and Applications FOI ITHEA, 2007, pp. 182–188. URL: http://oa.upm.es/2869/ (cit. on p. 41).

[173] F. Javier Gil et al. "Delimited Massively Parallel Algorithm Based on Rules Elimination for Application of Active Rules in Transition P Systems". In: *Information Technologies and Knowledge* 2.1 (2008), pp. 56–61. URL: http://hdl.handle.net/10525/268 (cit. on p. 41).

[174] A. Gutierrez et al. "Optimizing Membrane System Implementation with Multisets and Evolution Rules Compression". In: *Proceedings of the 8th Workshop on Membrane Computing*. 2007, pp. 345–362 (cit. on p. 41).

[175] F. J. G. Rubio, J. A. T. Cerbel, and L. F. Muñoz. "Fast Linear Algorithm for Active Rules Application in Transition P Systems". In: *Algorithmic and Mathematical Foundations of the Artificial Intelligence*. Ed. by K. Markov, K. Ivanova, and I. Mitov. Vol. Supple. International Book Series ?INFORMATION SCIENCE & COMPUTING? 1. Sofia, Bulgaria: Institute of Information Theories and Applications FOI ITHEA, Bulgaria, 2008, pp. 35–44. URL: http://oa.upm.es/2870/ (cit. on p. 41).

[176] F. J. G. Rubio et al. "Parallel algorithm for P Systems implementation in multiprocessors". In: *Proceedings of the Thirteenth International Symposium on Artificial Life and Robotics 2008 (AROB 13th'08)*. M. Sugisaka and H. Tanaka, 2008, pp. 0–0. URL: http://oa.upm.es/3161/ (cit. on p. 41).

[177] M. A. Martínez-del-Amor et al. "A new simulation algorithm for multienvironment probabilistic P systems". In: *Fifth International Conference on Bio-Inspired Computing: Theories and Applications, BIC-TA 2010, University of Hunan, Liverpool Hope University, Liverpool, United Kingdom / Changsha, China, September 8-10 and September 23-26, 2010*. Ed. by M. Gong et al. 2010, pp. 59–68. DOI: 10.1109/BICTA.2010.5645352. URL: https://doi.org/10.1109/BICTA.2010.5645352 (cit. on p. 42).

[178] M. A. Martínez-del-Amor et al. "DCBA: Simulating Population Dynamics P Systems with Proportional Object Distribution". In: *Membrane Computing - 13th International Conference, CMC 2012, Budapest, Hungary, August 28-31, 2012, Revised Selected Papers*. 2012, pp. 257–276. DOI: 10.1007/978-3-642-36751-9_18. URL: https://doi.org/10.1007/978-3-642-36751-9_18 (cit. on p. 42).

[179] M. A. Martínez-del-Amor et al. "Parallel simulation of Population Dynamics P systems: updates and roadmap". In: *Natural Computing* 15.4 (2016), pp. 565–573. DOI: 10.1007/s11047-016-9566-1. URL: https://doi.org/10.1007/s11047-016-9566-1 (cit. on p. 42).

[180] V. Nguyen, D. Kearney, and G. Gioiosa. "An extensible, maintainable and elegant approach to hardware source code generation in Reconfig-P". In: *J. Log. Algebr. Program.* 79.6 (2010), pp. 383–396. DOI: 10.1016/j.jlap.2010.03.013 (cit. on p. 44).

[181] Q. Fang et al. "Implementation and Research on EHW-based Digital Chip Using Handel-C Language". In: *Third International Conference on Natural Computation, ICNC 2007, Haikou, Hainan, China, 24-27 August 2007, Volume 5*. Ed. by J. Lei, J. Yao, and Q. Zhang. IEEE Computer Society, 2007, pp. 624–628. DOI: 10.1109/ICNC.2007.436 (cit. on p. 46).

[182] L. Middendorf and C. Bobda. "Declarative Programming with Handel-C". In: *Proceedings of the 2010 International Conference on Engineering of Reconfigurable Systems & Algorithms, ERSA 2010, July 12-15, 2010, Las Vegas Nevada, USA*. Ed. by T. P. Plaks et al. CSREA Press, 2010, pp. 151–164 (cit. on p. 46).

[183] L. Fernández et al. "A Hardware Circuit for Selecting Active Rules in Transition P Systems". In: *Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2005), 25-29 September 2005, Timisoara, Romania*. Ed. by D. Zaharie et al. IEEE Computer Society, 2005, pp. 415–418. DOI: 10.1109/SYNASC.2005.7 (cit. on p. 50).

[184] V. Martinez et al. "Hardware Implementation of a Bounded Algorithm for Application of Rules in a Transition P-system". In: *Symbolic and Numeric Algorithms for Scientific Computing, 2006. SYNASC'06. Eighth International Symposium on*. IEEE. 2006, pp. 343–349 (cit. on p. 50).

[185] A. Gutierrez et al. "Design of a Hardware Architecture Based on Microcontrollers for the Implementation of Membrane Systems". In: *8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2006), 26-29 September 2006, Timisoara, Romania*. Ed. by V. Negru et al. IEEE Computer Society, 2006, pp. 350–353. DOI: 10.1109/SYNASC.2006.29 (cit. on p. 50).

[186] V. Martinez et al. "HW Implementation of a Optimized Algorithm for the Application of Active Rules in a Transition P-system". In: *Information Theories and Applications* 14.4 (2007), pp. 324–331. URL: http://hdl.handle.net/10525/699 (cit. on p. 50).

[187] S. Alonso et al. "Main modules design for a HW implementation of massive parallelism in transition P-systems". In: *Artificial Life and Robotics* 13.1 (2008), pp. 107–111. ISSN: 1614-7456. DOI: 10.1007/s10015-008-0526-4. URL: https://doi.org/10.1007/s10015-008-0526-4 (cit. on p. 50).

[188] S. Alonso et al. "A circuit implementing massive parallelism in transition P systems". In: *Information Technologies and Knowledge* 2 (2008), pp. 35–42 (cit. on p. 50).

[189] S. M. G. Canaval, A. G. Rodriguez, and S. A. Villaverde. "Hardware Implementation of P Systems Using Microcontrollers. An Operating Environment for Implementing a Partially Parallel Distributed Architecture". In: *Symbolic and Numeric Algorithms for Scientific Computing, 2008. SYNASC'08. 10th International Symposium on*. IEEE. 2008, pp. 489–495 (cit. on p. 50).

[190] A. Gutiérrez et al. "Suitability of using microcontrollers in implementing new P-system communications architectures". In: *Artificial Life and Robotics* 13.1 (2008), pp. 102–106. ISSN: 1614-7456. DOI: 10.1007/s10015-008-0535-3. URL: https://doi.org/10.1007/s10015-008-0535-3 (cit. on p. 53).

[191] S. A. Villaverde et al. "A Circuit Implementing Massive Parallelism in Transition P Systems". In: *International Journal Information Technologies and Knowledge.* 2.1 (2008), pp. 35–42. URL: http://oa.upm.es/2194/ (cit. on p. 53).

[192] Z. Xu et al. "The stochastic loss of spikes in spiking neural P systems: Design and implementation of reliable arithmetic circuits". In: *Fundamenta Informaticae* 134.1-2 (2014), pp. 183–200 (cit. on p. 53).

[193] T. Ishdorj, O. Ochirbat, and C. Naimannaran. "A u-fluidic Biochip Design for Spiking Neural P Systems". In: *Int. J. Unconv. Comput.* 15.1-2 (2020), pp. 59–82. URL: https://www.oldcitypublishing.com/journals/ijuc-home/ijuc-issue-contents/ijuc-volume-15-number-1-2-2020/ijuc-15-1-2-p-59-82/ (cit. on p. 53).

[194] O. Ochirbat, T. Ishdorj, and G. Cichon. "An error-tolerant serial binary full-adder via a spiking neural P system using HP/LP basic neurons". In: *J. Membr. Comput.* 2.1 (2020), pp. 42–48. DOI: 10.1007/s41965-020-00033-3 (cit. on p. 53).

[195] G. H. Mealy. "A method for synthesizing sequential circuits". In: *The Bell System Technical Journal* 34.5 (1955), pp. 1045–1079 (cit. on pp. 55–57, 79, 91).

[196] C. E. Shannon. "A symbolic analysis of relay and switching circuits". In: *Electrical Engineering* 57.12 (1938), pp. 713–723. DOI: 10.1109/EE.1938.6431064 (cit. on p. 56).

[197] E. C. Nelson. "An algebraic theory for use in digital computer design". In: *Trans. I R E Prof. Group Electron. Comput.* 3.3 (1954), pp. 12–21. DOI: 10.1109/IREPGELC.1954.6499442 (cit. on p. 56).

[198] Z. Shang et al. "Reaction Systems and Synchronous Digital Circuits". In: *Molecules* 24.10 (May 21, 2019). ISSN: 1420-3049. DOI: 10.3390/molecules24101961. URL: https://www.mdpi.com/1420-3049/24/10/1961 (cit. on pp. 56, 69).

[199] Tutorialspoint. *Moore and Mealy Machines*. 2020. URL: https://www.tutorialspoint.com/automata_theory/moore_and_mealy_machines.htm (cit. on p. 56).

[200] E. M. Moore. "Gedanken-experiments on sequential machines". In: *Automata Studies*. Annals of Mathematics Studies. Princeton University Press, 1956, pp. 129–153 (cit. on pp. 56, 79, 91).

[201] Y. Crama and P. L. Hammer. *Boolean Functions - Theory, Algorithms, and Applications*. Vol. 142. Encyclopedia of mathematics and its applications. Cambridge University Press, 2011. ISBN: 978-0-521-84751-3. URL: http://www.cambridge.org/gb/knowledge/isbn/item6222210/?site_locale=en_GB (cit. on p. 58).

[202] M. Schliwa. *The Cytoskeleton: An Introductory Survey. An Introductory Survey*. Springer, 1986. ISBN: 978-3-7091-7669-6. DOI: 10.1007/978-3-7091-7667-2 (cit. on p. 62).

[203] E. Czeizler et al. "Quantitative Analysis of the Self-Assembly Strategies of Intermediate Filaments from Tetrameric Vimentin". In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 9.3 (2012), pp. 885–898. DOI: 10.1109/TCBB.2011.154 (cit. on p. 62).

[204] A. Mizera, E. Czeizler, and I. Petre. "Self-assembly Models of Variable Resolution". In: *Trans. Comp. Sys. Biology* 14 (2012), pp. 181–203. DOI: 10.1007/978-3-642-35524-0_8. URL: https://doi.org/10.1007/978-3-642-35524-0_8 (cit. on p. 62).

[205] H. Herrmann and U. Aebi. "Intermediate Filaments: Molecular Structure, Assembly Mechanism, and Integration Into Functionally Distinct Intracellular Scaffolds". In: *Annual Review of Biochemistry* 73.1 (2004), pp. 749–789. DOI: 10.1146/annurev.biochem.73.011303.073823. URL: https://doi.org/10.1146/annurev.biochem.73.011303.073823 (cit. on p. 62).

[206] R. I. Morimoto. "The heat shock response: Systems biology of proteotoxic stress in aging and disease". In: *Cold Spring Harbor Symposia on Quantitative Biology* 76 (2011), pp. 91–99. ISSN: 0091-7451. DOI: 10.1101/sqb.2012.76.010637 (cit. on p. 67).

[207] A. A. Asea and S. K. Calderwood. *Regulation of Heat Shock Protein Responses*. Springer, 2018. URL: https://link.springer.com/content/pdf/10.1007/978-3-319-74715-6.pdf (cit. on p. 67).

[208] T. R. Rieger, R. I. Morimoto, and V. Hatzimanikatis. "Mathematical Modeling of the Eukaryotic Heat-Shock Response: Dynamics of the hsp70 Promoter". In: *Biophysical Journal* 88.3 (2005), pp. 1646–1658. ISSN: 0006-3495. DOI: https://doi.org/10.1529/biophysj.104.055301 (cit. on p. 67).

[209] I. Petre et al. "A simple mass-action model for the eukaryotic heat shock response and its mathematical validation". In: *Nat. Comput.* 10.1 (2011), pp. 595–612. DOI: 10.1007/s11047-010-9216-y (cit. on p. 67).

[210] R.-J. Back, T.-O. Ishdorj, and I. Petre. "A Petri-net formalization of the heat shock response model". In: *Proceedings of COMPMOD 2008 Workshop on Computational Models for Cell Processes*. 2008, pp. 53–61. URL: https://www.researchgate.net/publication/31597900_A_Petri-net_formalization_of_the_heat_shock_response_model (cit. on p. 67).

[211] B. T. Cem Ünsalan. *Digital System Design with FPGA*. Ed. by M. Hill. McGraw Hill Education, 2017 (cit. on p. 73).

[212] D. E. Calbaza and Y. Savaria. "Jitter model of direct digital synthesis clock generators". In: *Proceedings of the 1999 International Symposium on Circuits and Systems, ISCAS 1999, Orlando, Florida, USA, May 30 - June 2, 1999*. IEEE, 1999, pp. 1–4. DOI: 10.1109/ISCAS.1999.777791 (cit. on p. 73).

[213] L. Xiu. "All digital FPGA-implementable time-average-frequency direct period synthesis for IoT applications". In: *IEEE International Symposium on Circuits and Systems, ISCAS 2017, Baltimore, MD, USA, May 28-31, 2017*. IEEE, 2017, pp. 1–4. DOI: 10.1109/ISCAS.2017.8050550 (cit. on p. 73).

[214] G. Darcheville, C. Voillequin, and J. Begueret. "Direct Digital Frequency Synthesis design methodology for optimized spurs / jitter performances". In: *25th IEEE International Conference on Electronics, Circuits and Systems, ICECS 2018, Bordeaux, France, December 9-12, 2018*. IEEE, 2018, pp. 733–736. DOI: 10.1109/ICECS.2018.8617927 (cit. on p. 73).

[215] E. Murphy and C. Slattery. *Ask The Application Engineer 33: All About Direct Digital Synthesis*. 2004. URL: https://www.analog.com/en/analog-dialogue/articles/all-about-direct-digital-synthesis.html# (cit. on p. 73).

[216] B. Han et al. *FPGA Design Tactics and Case Development*. 2nd ed. Pulishing House of Electrinics Industry, 2017 (cit. on p. 74).

[217] G. Păun and R. A. Păun. "Membrane Computing and Economics: Numerical P Systems". In: *Fundam. Inform.* 73.1-2 (2006), pp. 213–227. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi73-1-2-20 (cit. on pp. 79, 80).

[218] A. Pavel, O. Arsene, and C. Buiu. "Enzymatic numerical P systems - a new class of membrane computing systems". In: *Fifth International Conference on Bio-Inspired Computing: Theories and Applications, BIC-TA 2010, University of Hunan, Liverpool Hope University, Liverpool, United Kingdom / Changsha, China, September 8-10 and September 23-26, 2010*. IEEE, 2010, pp. 1331–1336. DOI: 10.1109/BICTA.2010.5645071 (cit. on pp. 79, 81, 102).

[219] Z. Zhang et al. "Numerical P systems with migrating variables". In: *Theor. Comput. Sci.* 641 (2016), pp. 85–108. DOI: 10.1016/j.tcs.2016.06.004 (cit. on p. 79).

[220] L. Pan et al. "Numerical P systems with production thresholds". In: *Theor. Comput. Sci.* 673 (2017), pp. 30–41. DOI: 10.1016/j.tcs.2017.02.026 (cit. on p. 79).

[221] Z. Zhang et al. "Universal enzymatic numerical P systems with small number of enzymatic variables". In: *SCIENCE CHINA Information Sciences* 61.9 (2018), 092103:1–092103:12. DOI: 10.1007/s11432-017-9103-5 (cit. on p. 79).

[222] Z. Zhang, T. Wu, and L. Pan. "On String Languages Generated by Sequential Numerical P Systems". In: *Fundam. Inform.* 145.4 (2016), pp. 485–509. DOI: 10.3233/FI-2016-1372 (cit. on p. 79).

[223] L. Pan et al. "Numerical P systems with production thresholds". In: *Theor. Comput. Sci.* 673 (2017), pp. 30–41. DOI: 10.1016/j.tcs.2017.02.026 (cit. on p. 79).

[224] L. Pan et al. "Four Recent Research Topics on Numerical and Spiking Neural P Systems". In: *Romanian Journal of Information Science and Technology* 19.1-2 (2016), pp. 5–16 (cit. on p. 79).

[225] C. I. Vasile, A. B. Pavel, and I. Dumitrache. "Universality of Enzymatic Numerical P systems". In: *Int. J. Comput. Math.* 90.4 (2013), pp. 869–879. DOI: 10.1080/00207160.2012.748897 (cit. on p. 79).

[226] A. Leporati et al. "Enzymatic Numerical P Systems Using Elementary Arithmetic Operations". In: *Membrane Computing - 14th International Conference, CMC 2013, Chişinău, Republic of Moldova, August 20-23, 2013, Revised Selected Papers*. Ed. by A. Alhazov et al. Vol. 8340. Lecture Notes in Computer Science. Springer, 2013, pp. 249–264. DOI: 10.1007/978-3-642-54239-8_18. URL: https://doi.org/10.1007/978-3-642-54239-8_18 (cit. on p. 79).

[227] Z. Zhang, Y. Su, and L. Pan. "The computational power of enzymatic numerical P systems working in the sequential mode". In: *Theor. Comput. Sci.* 724 (2018), pp. 3–12. DOI: 10.1016/j.tcs.2017.12.016 (cit. on p. 79).

[228] S. Pang et al. "A Parallel Bioinspired Framework for Numerical Calculations Using Enzymatic P System With an Enzymatic Environment". In: *IEEE Access* 6 (2018), pp. 65548–65556. DOI: 10.1109/ACCESS.2018.2876364 (cit. on p. 79).

[229] S. Maeda and A. Fujiwara. "Enzymatic numerical P systems for basic operations and sorting". In: *2014 Joint 7th International Conference on Soft Computing and Intelligent Systems (SCIS) and 15th International Symposium on Advanced Intelligent Systems (ISIS), Kita-Kyushu, Japan, December 3-6, 2014*. IEEE, 2014, pp. 1333–1338. DOI: 10.1109/SCIS-ISIS.2014.7044708 (cit. on p. 79).

[230] C. I. Vasile, A. B. Pavel, and I. Dumitrache. "Improving the Universality Results of Enzymatic Numerical P Systems". In: *Proceedings of the Tenth Brainstorming Week on Membrane Computing*. Vol. 2. Feb. 2012, pp. 207–214. URL: https://idus.us.es/xmlui/handle/11441/34150 (cit. on p. 79).

[231] D. Llorente-Rivera and M. A. Gutiérrez-Naranjo. "The Pole Balancing Problem with Enzymatic Numerical P Systems". In: *Proceedings of the Thirteenth Brainstorming Week on Membrane Computing*. Feb. 2015, pp. 195–206 (cit. on p. 79).

[232] C. E. Shannon. "The synthesis of two-terminal switching circuits". In: *The Bell System Technical Journal* 28.1 (1949), pp. 59–98. ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1949.tb03624.x (cit. on pp. 79, 91).

[233] G. Berry. "Esterel and Jazz: Two Synchronous Languages for Circuit Design (Abstract)". In: *Correct Hardware Design and Verification Methods, 10th IFIP WG 10.5 Advanced Research Working Conference, CHARME '99, Bad Herrenalb, Germany, September 27-29, 1999, Proceedings*. Ed. by L. Pierre and T. Kropf. Vol. 1703. Lecture Notes in Computer Science. Springer, 1999, p. 1. DOI: 10.1007/3-540-48153-2_1. URL: https://doi.org/10.1007/3-540-48153-2_1 (cit. on p. 79).

[234] G. Berry. "The foundations of Esterel". In: *Proof, Language, and Interaction, Essays in Honour of Robin Milner*. Ed. by G. D. Plotkin, C. Stirling, and M. Tofte. The MIT Press, 2000, pp. 425–454 (cit. on p. 79).

[235] B. Rajan and R. K. Shyamasundar. "Modeling VHDL in Multiclock ESTEREL". In: *13th International Conference on VLSI Design (VLSI Design 2000), 4-7 January 2000, Calcutta, India*. IEEE Computer Society, 2000, pp. 76–83. DOI: 10.1109/ICVD.2000.812588 (cit. on p. 79).

[236] J. Bergerand. "LUSTRE : un langage déclaratif pour le temps réel. (LUSTRE: a real time declarative language)". PhD thesis. Grenoble Institute of Technology, France, 1986. URL: https://tel.archives-ouvertes.fr/tel-00320006 (cit. on p. 79).

[237] K. Qian et al. "LUSTRE: An Interactive System for Entity Structured Representation and Variant Generation". In: *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*. IEEE Computer Society, 2018, pp. 1613–1616. DOI: 10.1109/ICDE.2018.00189 (cit. on p. 79).

[238] C. Buiu, C. I. Vasile, and O. Arsene. "Development of membrane controllers for mobile robots". In: *Inf. Sci.* 187 (2012), pp. 33–51. DOI: 10.1016/j.ins.2011.10.007 (cit. on pp. 80, 88, 101, 102, 112).

[239] A. B. Pavel, C. I. Vasile, and I. Dumitrache. "Robot Localization Implemented with Enzymatic Numerical P Systems". In: *Biomimetic and Biohybrid Systems - First International Conference, Living Machines 2012, Barcelona, Spain, July 9-12, 2012. Proceedings*. Ed. by T. J. Prescott et al. Vol. 7375. Lecture Notes in Computer Science. Springer, 2012, pp. 204–215. DOI: 10.1007/978-3-642-31525-1_18. URL: https://doi.org/10.1007/978-3-642-31525-1_18 (cit. on pp. 81, 102).

[240] A. Leporati et al. "Improved Universality Results for Parallel Enzymatic Numerical P Systems". In: *International Journal of Unconventional Computing* 9.5-6 (2013), pp. 385–404. URL: http://www.oldcitypublishing.com/journals/ijuc-home/ijuc-issue-contents/ijuc-volume-9-number-5-6-2013/ijuc-9-5-6-p-385-404/ (cit. on p. 81).

[241] F. R. Villatoro and J. I. Ramos. "On the method of modified equations. I: Asymptotic analysis of the Euler forward difference method". In: *Appl. Math. Comput.* 103.2-3 (1999), pp. 111–139. DOI: 10.1016/S0096-3003(98)10031-0 (cit. on p. 84).

[242] X. Shen, X. Cheng, and K. Liang. "Deep Euler method: solving ODEs by approximating the local truncation error of the Euler method". In: *CoRR* abs/2003.09573 (2020). arXiv: 2003.09573. URL: https://arxiv.org/abs/2003.09573 (cit. on p. 84).

[243] A. G. Florea and C. Buiu. *GitHub—PeP: (Enzymatic) Numerical P System simulator*. URL: https://github.com/andrei91ro/pep (cit. on pp. 88, 95, 107, 122).

[244] Y. Liu and B. Xin. "Numerical Solutions of a Fractional Predator-Prey System". In: *Advances in Difference Equations* 2011 (2011) (cit. on p. 89).

[245] D.-H. Go. "Three Essays in Economics of Prey-Predator Relation". PhD thesis. Utah State University, 2018. URL: https://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=8138&context=etd (cit. on p. 89).

[246] J. Cavanagh. *Sequential Logic and Verilog HDL Fundamentals*. CRC Press, 2016 (cit. on p. 95).

[247] *Xilinx CORDIC v6.0 LogiCORE IP Product Guide*. 2017. URL: https://www.xilinx.com/support/documentation/ip_documentation/cordic/v6_0/pg105-cordic.pdf (cit. on p. 95).

[248] I. Pérez-Hurtado et al. "P-Lingua in two steps: flexibility and efficiency". In: *Journal of Membrane Computing* 1.2 (2019), pp. 93–102. ISSN: 2523-8906. DOI: 10.1007/s41965-019-00010-5. URL: https://doi.org/10.1007/s41965-019-00010-5 (cit. on p. 95).

[249] S. N. Krishna. "An Overview of Membrane Computing". In: *Distributed Computing and Internet Technology - 7th International Conference, ICDCIT 2011, Bhubaneshwar, India, February 9-12, 2011*. Vol. 6536. Lecture Notes in Computer Science. Springer, 2011, pp. 1–14. DOI: 10.1007/978-3-642-19056-8_1. URL: https://doi.org/10.1007/978-3-642-19056-8_1 (cit. on p. 101).

[250] T. Wang et al. "Fault Diagnosis of Electric Power Systems Based on Fuzzy Reasoning Spiking Neural P Systems". In: *IEEE Transactions on Power Systems* 30.3 (May 2015), pp. 1182–1194. ISSN: 0885-8950. DOI: 10.1109/TPWRS.2014.2347699 (cit. on p. 101).

[251] K. Huang et al. "Fault Classification of Power Transmission Lines Using Fuzzy Reasoning Spiking Neural P Systems". In: *Bio-inspired Computing – Theories and Applications* (Jan. 1, 2016). DOI: 10.1007/978-981-10-3611-8_12. URL: http://dx.doi.org/10.1007/978-981-10-3611-8_12 (cit. on p. 101).

[252] H. Peng et al. "Fault Diagnosis of Power Systems Using Intuitionistic Fuzzy Spiking Neural P Systems". In: *IEEE Transactions on Smart Grid* 9.5 (Sept. 2018), pp. 4777–4784. ISSN: 1949-3053. DOI: 10.1109/TSG.2017.2670602 (cit. on p. 101).

[253] H. Rong et al. "A novel approach for detecting fault lines in a small current grounding system using fuzzy reasoning spiking neural P systems". In: *International Journal of Computers, Communications & Control* 13.4 (2018), pp. 521–536 (cit. on p. 101).

[254] H. Rong et al. "Automatic Implementation of Fuzzy Reasoning Spiking Neural P Systems for Diagnosing Faults in Complex Power Systems". In: *Complexity* 2019 (2019), Article ID 2635714, 16 pages. DOI: 10.1155/2019/2635714 (cit. on p. 101).

[255] E. Sánchez-Karhunen and L. Valencia-Cabrera. "Modelling complex market interactions using PDP systems". In: *Journal of Membrane Computing* 1.1 (2019), pp. 40–51. ISSN: 2523-8906. DOI: 10.1007/s41965-019-00008-z. URL: https://doi.org/10.1007/s41965-019-00008-z (cit. on p. 101).

[256] G. Zhang et al. "A hybrid approach based on differential evolution and tissue membrane systems for solving constrained manufacturing parameter optimization problems". In: *Applied Soft Computing* 13.3 (2013), pp. 1528–1542. DOI: 10.1016/j.asoc.2012.05.032 (cit. on p. 101).

[257] G. Zhang et al. "An Optimization Spiking Neural P System for Approximately Solving Combinatorial Optimization Problems". In: *International Journal of Neural Systems* 24.5 (2014), Article No. 1440006, 16 pages. DOI: 10.1142/S0129065714400061 (cit. on p. 101).

[258] G. Zhang et al. "Evolutionary membrane computing: A comprehensive survey and new results". In: *Information Sciences* 279 (2014), pp. 528–551. DOI: 10.1016/j.ins.2014.04.007 (cit. on p. 101).

[259] X. Wang et al. "Design and implementation of membrane controllers for trajectory tracking of nonholonomic wheeled mobile robots". In: *Integrated Computer-Aided Engineering* 23.1 (2016), pp. 15–30. DOI: 10.3233/ICA-150503 (cit. on pp. 101, 102).

[260] X. Wang et al. "Multi-behaviors coordination controller design with enzymatic numerical P systems for robots". In: *Integrated Computer-Aided Engineering* 27 (2020), in press (cit. on pp. 101, 102).

[261] A. G. Florea and C. Buiu. "Modelling multi-robot interactions using a generic controller based on numerical P systems and ROS". In: *2017 9th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*. 2017, pp. 1–6. DOI: 10.1109/ECAI.2017.8166411 (cit. on p. 102).

[262] X. Wang et al. "Multi-behaviors Coordination Controller Design with Enzymatic Numerical P systems for Autonomous Mobile Robots in Unknown Environments". In: *Proceedings of the Asian Branch of International Conference on Membrane Computing (ACMC2018)*. Ed. by M. J. Dinneen and R. Nicolescu. Vol. 530. Centre for Discrete Mathematics and Theoretical Computer Science, Auckland, New Zealand, 2018, pp. 257–287 (cit. on p. 102).

[263] M. García-Quismondo, L. F. Macías-Ramos, and M. J. Pérez-Jiménez. "Implementing Enzymatic Numerical P Systems for AI Applications by Means of Graphic Processing Units". In: *Beyond Artificial Intelligence: Contemplations, Expectations, Applications*. Berlin, Heidelberg: Springer, 2013, pp. 137–159. ISBN: 978-3-642-34422-0. DOI: 10.1007/978-3-642-34422-0_10. URL: https://doi.org/10.1007/978-3-642-34422-0_10 (cit. on p. 102).

[264] A. B. Pavel and C. Buiu. "Using enzymatic numerical P systems for modeling mobile robot controllers". In: *Natural Computing* 11.3 (2012), pp. 387–393. DOI: 10.1007/s11047-011-9286-5 (cit. on p. 102).

[265] C. I. Vasile et al. "Implementing Obstacle Avoidance and Follower Behaviors on Koala Robots Using Numerical P Systems". In: *Tenth Brainstorming Week on Membrane Computing*. 2012, pp. 215–227 (cit. on p. 102).

[266] X. Wang et al. "A Modified Membrane Inspired Algorithm Based on Particle Swarm Optimization for Mobile Robot Path Planning". In: *International Journal of Computers* 6 (Oct. 2015), pp. 732–745 (cit. on p. 102).

[267] J. Yuan et al. "A Resolution-Free Parallel Algorithm for Image Edge Detection within the Framework of Enzymatic Numerical P Systems". In: *Molecules* (Mar. 2019). URL: https://www.mdpi.com/1420-3049/24/7/1235 (cit. on p. 102).

[268] Omron. *Pioneer 3 Operations Manual*. 2017 (cit. on pp. 112, 116).

[269] Z. Huang et al. "Mobile Robot Membrane Controller Design with Enzymatic Numerical P Systems for ObstacleAvoidance Behavior ". In: *Computer Systems and Applications* 28.7 (2019), pp. 17–25. DOI: 10.15888/j.cnki.csa.006976 (cit. on p. 113).

[270] I. Pérez-Hurtado et al. "Simulation of Rapidly-Exploring Random Trees in Membrane Computing with P-Lingua and Automatic Programming". In: *Int. J. Comput. Commun. Control* 13.6 (2018), pp. 1007–1031. DOI: 10.15837/ijccc.2018.6.3370 (cit. on p. 121).

[271] M. García-Quismondo et al. "P-Lingua 2.0: A software framework for cell-like P systems". In: *Int. J. Comput. Commun. Control* 4.3 (2009), pp. 234–243. DOI: 10.15837/ijccc.2009.3.2431 (cit. on p. 121).

[272] G. S. Malik et al. "FPGA based hierarchical architecture for parallelizing RRT". In: *Proceedings of the 2015 Conference on Advances In Robotics, AIR 2015, Goa, India, July 2-4, 2015*. ACM, 2015, 12:1–12:6. DOI: 10.1145/2783449.2783461 (cit. on p. 121).

[273] G. Malik et al. "FPGA based hybrid architecture for parallelizing RRT". In: *Computer Research Repository* abs/1607.05704 (2016). arXiv: 1607.05704. URL: http://arxiv.org/abs/1607.05704 (cit. on p. 121).

[274] G. S. Malik. "FPGA based massively parallel architectures for super fast path planning via Rapidly Exploring Random Trees (RRT)". MA thesis. International Institute of Information Technology, Hyderabad, 2016 (cit. on p. 121).

[275] S. Xiao, N. Bergmann, and A. Postula. "Parallel RRT$*$ architecture design for motion planning". In: *Proc. 27th Int. Conf. Field Programmable Logic and Applications (FPL)*. 2017, pp. 1–4 (cit. on p. 121).

[276] S. M. LaValle. *Rapidly-Exploring Random Trees: A New Tool for Path Planning*. Tech. rep. Department of Computer Science, Iowa State University, 1998 (cit. on p. 121).

[277] S. M. Lavalle and J. J. Kuffner. "Rapidly-Exploring Random Trees: Progress and Prospects". In: *Proceedings IEEE International Conference on Robotics and Automation*. 2000, pp. 995–1001 (cit. on p. 121).

[278] R. Pepy and A. Lambert. "Safe Path Planning in an Uncertain-Configuration Space using RRT". In: *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*. 2006, pp. 5376–5381 (cit. on p. 121).

[279] I. Aguinaga, D. Borro, and L. Matey. "Parallel RRT-based path planning for selective disassembly planning". In: *International Journal of Advanced Manufacturing Technology* 36 (2008), pp. 1221–1233. ISSN: 0268-3768. DOI: 10.1007/s00170-007-0930-2 (cit. on p. 121).

[280] L. Jaillet, J. Cortes, and T. Simeon. "Transition-based RRT for path planning in continuous cost spaces". In: *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*. 2008, pp. 2145–2150 (cit. on p. 121).

[281] D. Devaurs, T. Siméon, and J. Cortés. "A multi-tree extension of the transition-based RRT: Application to ordering-and-pathfinding problems in continuous cost spaces". In: *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*. 2014, pp. 2991–2996 (cit. on p. 121).

[282] H. Umari and S. Mukhopadhyay. "Autonomous robotic exploration based on multiple rapidly-exploring randomized trees". In: *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*. 2017, pp. 1396–1402 (cit. on p. 121).

[283] J. Y. F. Tong, D. Nagle, and R. A. Rutenbar. "Reducing power by optimizing the necessary precision/range of floating-point arithmetic". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 8.3 (2000), pp. 273–286. DOI: 10.1109/92.845894 (cit. on p. 129).

[284] C. F. Fang, T. Chen, and R. A. Rutenbar. "Lightweight Floating-Point Arithmetic: Case Study of Inverse Discrete Cosine Transform". In: *EURASIP Journal on Advances in Signal Processing* 2002.9 (2002), pp. 879–892. DOI: 10.1155/S1110865702205090 (cit. on p. 129).

[285] Ryszard. *Origin of Quake3's Fast InvSqrt()*. 2007. URL: https://www.beyond3d.com/content/articles/8/ (cit. on p. 131).

[286] C. Lomont. *Fast Inverse Square Root*. Research rep. Department of Mathematics, Purdue University, 2003 (cit. on pp. 131, 132).

[287] K. H. Tsoi, K. H. Leung, and P. H. W. Leong. "Compact FPGA-based True and Pseudo Random Number Generators". In: *11th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2003), 8-11 April 2003, Napa, CA, USA, Proceedings*. IEEE Computer Society, 2003, pp. 51–61. DOI: 10.1109/FPGA.2003.1227241 (cit. on p. 132).

[288] P. Kohlbrenner and K. Gaj. "An embedded true random number generator for FPGAs". In: *Proceedings of the ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays, FPGA 2004, Monterey, California, USA, February 22-24, 2004*. Ed. by R. Tessier and H. Schmit. ACM, 2004, pp. 71–78. DOI: 10.1145/968280.968292 (cit. on p. 132).

[289] M. Majzoobi, F. Koushanfar, and S. Devadas. "FPGA-Based True Random Number Generation Using Circuit Metastability with Adaptive Feedback Control". In: *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*. Ed. by B. Preneel and T. Takagi. Vol. 6917. Lecture Notes in Computer Science. Springer, 2011, pp. 17–32. DOI: 10.1007/978-3-642-23951-9_2. URL: https://doi.org/10.1007/978-3-642-23951-9_2 (cit. on p. 132).

[290] Xilinx. *Virtex-7 FPGA VC707 Evaluation Kit*. 2020. URL: https://www.xilinx.com/products/boards-and-kits/ek-v7-vc707-g.html#overview (cit. on p. 135).

Publications by the author

1. Gexiang Zhang, **Zeyi Shang**, Sergey Verlan, Miguel A. Martínez-del-Amor, Chengxun Yuan, Luis Valencia-Cabrera, Mario J. Pérez-Jiménez. An Overview of Hardware Implementation of Membrane Computing Models. *ACM Computing Surveys*, 2020, 53(4), DOI:10.1145/3402456.

2. **Zeyi Shang**, Sergey Verlan, Ion Petre, Gexiang Zhang. Reaction Systems and Synchronous Digital Circuits. *Molecules*, 2019, 24(10), DOI:10.3390/molecules24101961.

3. **Zeyi Shang**, Sergey Verlan, Gexiang Zhang. FPGA Implementations of Numerical P Systems. *International of Journal of Unconventional Computing*, 2020. (to appear)

4. **Zeyi Shang**, Sergey Verlan, Gexiang Zhang, Miguel A. Martinez-del-Amor, Luis Valencia-Cabrera, Mario J. Pérez-Jiménez. An Overview of Hardware Implementations of P Systems[C]. In *Pre-proceedings of the 6th Asian Branch of International Conference on Membrane Computing*, September 21-25, 2017, Chengdu, China, pp. 487-526.

5. **Zeyi Shang**, Sergey Verlan, Gexiang Zhang. Hardware Implementation of Numerical P Systems. In *Pre-proceedings of the 20th International Conference on Membrane Computing*, CMC20, August 5-8, 2019, Curtea de Arges, Romania, pp. 463-474.

6. **Zeyi Shang**, Sergey Verlan, Gexiang Zhang, Ignacio Pérez-Hurtado. FPGA Implementation of Robot Obstacle Avoidance Controller based on Enzymatic Numerical P Systems. In *Pre-proceedings of the 8th Asian Branch of International Conference on Membrane Computing*, November 14-17, 2019, Xiamen, China, pp. 184-214.

7. **Zeyi Shang**, Ignacio Pérez-Hurtado, Gexiang Zhang, Sergey Verlan. FPGA Architecture for Generalized Numerical P System Arranged Rapid-exploring Random Tree Algorithm. In *Pre-proceedings of the 2020 International Conference on Membrane Computing*, September 14-18, 2020, Ulaanbaatar, Mongolia. (to appear)