EFFICIENT RUNTIME MANAGEMENT OF RECONFIGURABLE HARDWARE RESOURCES

THOMAS MARCONI

EFFICIENT RUNTIME MANAGEMENT OF RECONFIGURABLE HARDWARE RESOURCES THOMAS MARCONI

Thomas Marconi

Efficient Runtime Management of Reconfigurable Hardware Resources

Efficient Runtime Management of Reconfigurable Hardware Resources

---

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof.ir. K.C.A.M Luyben,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op

woensdag 29 Juni 2011 om 12:30 uur

door

Thomas MARCONI

Magister Teknik in Electrical Engineering
Bandung Institute of Technology, Indonesië
geboren te Belinyu, Indonesië

Dit proefschrift is goedgekeurd door de promotor:
Prof. dr. ir. H.J. Sips

Copromotor:
Dr. ir. G.N. Gaydadjiev

Samenstelling promotiecommissie:

Rector Magnificus,	voorzitter
Prof. dr. ir. H.J. Sips,	Technische Universiteit Delft, promotor
Dr. ir. G.N. Gaydadjiev,	Technische Universiteit Delft, copromotor
Prof. dr. C.I.M. Beenakker,	Technische Universiteit Delft
Prof. dr. W. Najjar,	University of California, USA
Prof. dr. J.H. Takala,	Tampere University of Technology, Finland
Prof. dr. D.N.Pnevmatikatos,	Technical University of Crete, Griekenland
Dr. K.L.M. Bertels,	Technische Universiteit Delft
Prof. dr. P.M. Sarro,	Technische Universiteit Delft, reservelid

CIP-DATA KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Thomas Marconi

Efficient Runtime Management of Reconfigurable Hardware Resources
Delft: TU Delft, Faculty of Elektrotechniek, Wiskunde en Informatica - III
Thesis Technische Universiteit Delft. – With ref. –

Met samenvatting in het Nederlands.

ISBN 978-90-72298-20-1

Subject headings: runtime reconfigurable systems, partially reconfigurable devices, scheduling, placement, low power, reconfiguration, reconfigurable computing.

Copyright © 2011 Thomas Marconi
All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without permission of the author.

Printed in The Netherlands

To Jesus Christ: my Lord, my Creator, my Motivator, my Savior, my Friend, my Healer, my Mentor, my Teacher, and my Redeemer.

Efficient Runtime Management of Reconfigurable Hardware Resources

Thomas Marconi

Abstract

R**untime** reconfigurable systems built upon devices with partial reconfiguration can provide reduction in overall hardware area, power efficiency, and economic cost in addition to the performance improvements due to better customization. However, the users of such systems have to be able to afford some additional costs compared to hardwired application specific circuits. More precisely reconfigurable devices have higher power consumption, occupy larger silicon area and operate at lower speeds. Higher power consumption requires additional packaging cost, shortens chip lifetimes, requires expensive cooling systems, decreases system reliability and prohibits battery operation. The less efficient usage of silicon real estate is usually compensated by the runtime hardware reconfiguration and functional units relocation. The available configuration data paths, however, have limited bandwidth that introduces overheads that may eclipse the dynamic reconfiguration benefits. In this dissertation, we address three major problems related to hardware resources runtime management: efficient online hardware task scheduling and placement, power consumption reduction and reconfiguration overhead minimization. Since hardware tasks are allocated and deallocated dynamically at runtime, the reconfigurable fabric can suffer of fragmentation. This can lead to the undesirable situation that tasks cannot be allocated even if there would be sufficient free area available. As a result, the overall system performance is degraded. Therefore, efficient hardware management of resources is very important. To manage hardware resources efficiently, we propose novel online hardware task scheduling and placement algorithms on partially reconfigurable devices with higher quality and faster execution compared to related proposals. To cope with the high power consumption in field programmable devices, we propose a novel logic element with lower power consumption compared to current approaches. To reduce runtime overhead, we augment the FPGA configuration circuit architecture and allow faster reconfiguration and relocation compared to current reconfigurable devices.

Acknowledgements

It will not be enough space to name all people who have used their various talents to help me finishing this work. Without all these wonderful men and women created by Almighty God, I could not finish this work. Thank God for His endless provision.

Although Stamatis Vassiliadis is not in this world anymore, his influence is never ended. I'm thankful to Stamatis for giving me the opportunity to work on the hArtes project and also for his short guidance before he left his last mark to many young scientists, especially in CE Lab of TU Delft.

I owe my deepest gratitude to George Gaydadjiev for supervising me. Although it was painful, his high quality expectation trained me to learn more and work harder for moving forward beyond my limitations. He showed me a real example of being a scientist. Although he was extremely busy, he invested a lot of his precious time for me. He helped me to fix this work even during the ICCD conference. He helped me to deal with all challenges I faced.

I would like to express the deepest appreciation to my other supervisor, Koen Bertels. He not only supervised me, but also he motivated me. This combination made me strong to keep doing my research despite many challenges. He demonstrated me a good example of being a teacher. He spent a lot of his time with me during my research even in his busiest moment. Moreover, without his phone calls, I would not be in Delft.

I'm thankful to my promotor, Prof. Henk Sips, for giving me the opportunity to officially finish this thesis. He significantly helped me to improve the quality of this thesis and the propositions. Without him, I would not be a doctor.

I would like to thank all of my doctoral committee: Prof. K.C.A.M. Luyben, Prof. H.J. Sips, Dr. G.N. Gaydadjiev, Dr. K.L.M. Bertels, Prof. C.I.M. Beenakker, Prof. W. Najjar, Prof. J.H. Takala, Prof. D.N. Pnevmatikatos, and Prof. P.M. Sarro. They gave their precious time for improving the quality of this thesis.

I am grateful to work with many of my colleagues. I'm thankful to Yi Lu for working together in the area of online task scheduling and placement. I am indebted to Dimitris for working together in dealing with low power designs for reconfigurable devices. It is a pleasure to work together with Jae in investigating fast configuration circuit architecture. I am also happy to work together with Zubair for Recursive Variable Expansion. I am also indebted to Roel for the Dutch translation of the abstract and also for proofreading this work. I'm thankful to Carlo for providing the Latex template and printing this thesis.

I thank all members of hArtes project for financing this work. It's a great opportunity for me to work together with many professionals in this large project. I would like to thank all of my room mates: Yi Lu, Dimitris, and Fakhar. Thanks for being with me, chatting with me, and helping me during my research. They made our office enjoyable. Doing research at TU Delft also needs administrative and technical assistance. I would like to thank Lidwina, Marjo, Bert, Monique, Erik, and Eef for helping me with administrative and technical stuff during my work at TU Delft.

I will never forget a person that helped me and my family when the first time we arrived in the Netherlands. Thank you very much Oom Hong Tjoen. He helped us to get our accommodation before we came to Delft. He guided us to arrive safely in Delft and picked us up to our first place to live in Delft.

I also would like to thank to all my teachers who taught me about all subjects I need and especially about how to live a better life for being a blessing to many people around me. Thank you very much to all men/women of God who always pray for me to make me strong for my journey in this wonderful life.

Without my parents, I would not be here. I'm grateful to my parents who raised me, who loved me, who taught me, who trained me, and who supported me. They have invested whole of their life to me. Thank my dearest Papa and Mama. Your names are always in my heart.

Life is not just doing a research, life is more than that. My life is more beautiful when I'm with my family. Grace Elizabeth, my dearest wife, thank you for loving me, for supporting me, for helping me, and for being with me. Your love inspired me to be a better husband. Joseph Abraham Thomas, my dearest son, thank you for playing together. Your humorous character refreshed me after working whole day. Eliana Marvella Thomas, my dearest daughter, thank you for supporting me with your special way. By carrying you, I felt peaceful.

Thomas Marconi Delft, The Netherlands, June 2011

Table of contents

Abstract . . . . **i**

Acknowledgments . . . . **iii**

List of Tables . . . . **ix**

List of Figures . . . . **xi**

List of Acronyms and Symbols . . . . **xv**

1 Introduction **1**
  1.1 Problem Overview 2
  1.2 Research Questions and Main Contributions 10
  1.3 Dissertation Organization 11

2 Related Work **13**
  2.1 Online Hardware Task Placement Algorithms 13
  2.2 Online Task Scheduling and Placement 21
  2.3 Low Power Techniques for Reconfigurable Devices 22
  2.4 Reconfiguration Overhead Reduction Techniques 29
  2.5 Summary 33

3 Online Hardware Task Placement Algorithms **35**
  3.1 Intelligent Merging Algorithm 36
    3.1.1 Basic Idea of the Intelligent Merging Strategy 36
    3.1.2 The Merging Only if Needed Technique 36
    3.1.3 The Partial Merging Technique 37
    3.1.4 Direct Combine and Combine Before Placing 38
    3.1.5 Intelligent Merging Algorithm 39
    3.1.6 Evaluation 40
  3.2 Proposed Quad-Corner Algorithm 45

3.2.1 Basic Idea of Quad-Corner Strategy 45
3.2.2 Two-dimensional Reconfigurable Device 46
3.2.3 Task Types 46
3.2.4 Initial Placement Positions 47
3.2.5 Data Structures 48
3.2.6 Searching Sequences for Placement 48
3.2.7 The Algorithm 48
3.2.8 Evaluation 49
3.3 Summary 54

4 Online Hardware Task Scheduling and Placement Algorithms 55
4.1 Intelligent Stuffing Algorithm for 1D Area Model 56
4.1.1 1D Area Scheduling and Placement Problems 56
4.1.2 Intelligent Stuffing Algorithm Main Properties 57
4.1.3 The Proposed Algorithm 57
4.1.4 Evaluation 60
4.2 Proposed 3D Compaction Algorithm for 2D Area Model . . . 61
4.2.1 Problem of Scheduling and Placement on 2D Area Model 61
4.2.2 Blocking-Aware Algorithm Main Idea 62
4.2.3 3D Total Contiguous Surface (3DTCS) Heuristic . . . 63
4.2.4 The 3D Compaction (3DC) Algorithm 65
4.2.5 Evaluation 69
4.3 Summary 74

5 Low Power Logic Element for FPDs 75
5.1 Introduction 75
5.2 The Proposed Logic Element 77
5.3 Transistor-level Circuit Evaluation 83
5.3.1 Experimental Setup 83
5.3.2 Experimental Results 86
5.4 Evaluation using a Real CAD Tool on a Real FPGA 89
5.4.1 Experimental Setup 89
5.4.2 Experimental Results 92
5.5 Summary 96

6 Improved Configuration Circuit Architecture for FPGAs 99
6.1 Introduction 99
6.2 Configuration Circuit Architecture 100

6.3 Evaluation 107
6.4 Summary 109

7 Conclusions and Future Work 111
7.1 Main Contributions 111
7.2 Open Issues 112

Bibliography 115

List of Publications 137

Samenvatting 141

Curriculum Vitae 143

List of Tables

1.1 Virtex FPGAs 5

3.1 Simulated task sets (W:task width, H:task height, Lt:task lifetime) 41

3.2 Some examples of implemented hardware tasks (et_i for 100 operations, rt_i at 100 MHz) 50

4.1 Obtained results using 100,000 task sets (TWA:total wasted area, TST:total schedule time, TRT:total response time and TAT:total algorithm execution time) 60

4.2 Computations of horizontal contiguous surfaces for positions in Figure 4.6(1)-(14) 65

4.3 Computations of vertical contiguous surfaces with scheduled tasks for positions in Figure 4.7(a)(1)-(16) 65

4.4 Computations of vertical contiguous surfaces with the FPGA boundary for positions in Figure 4.7(b)(1)-(8) 67

4.5 Time complexity analysis of 3D Compaction algorithm 69

5.1 The true table of $\mathtt{CLCG_1(conv)}$ 82

5.2 The true table of $\mathtt{CLCG_1(our)}$ 82

5.3 The MCNC benchmark circuits 83

5.4 Experimental results of Logic, Clock, and Interconnect Power (μW) 87

5.5 Experimental results of Dynamic, Static, and Total Power (μW) 87

5.6 Experimental results of power consumption at 50 MHz 93

5.7 Experimental results of area and maximum clock frequency . . 94
5.8 Power reduction at 100, 150, and 200 MHz 94
5.9 Comparison to clock gating solutions 96

6.1 Speedup (S) (times) and bitstream size (BSS) reduction (R) (%) compared to the conventional FPGA (h and w are in number of occupied CLBs) 108
6.2 Area overhead 109

List of Figures

1.1 Running multiple applications concurrently on a partially reconfigurable device 2
1.2 System Model (OS: operating system, P: placer, S: scheduler) 7
1.3 System at Design Time 8
1.4 System at Runtime 9

2.1 KAMER vs KNER 14
2.2 A rejection of an arriving task due to its previous wrong splitting decision 15
2.3 Area matrix of the staircase algorithm 16
2.4 A staircase 17
2.5 Area matrix of SLA 17
2.6 Fragmentation matrix of CF 18
2.7 FAP 20
2.8 Stuffing vs Classified Stuffing 21
2.9 Clock gating 24
2.10 A partially reconfigurable FPGA 30

3.1 MON technique 36
3.2 PM technique 37
3.3 DC technique 38
3.4 Pseudocode of Intelligent Merging algorithm 40
3.5 Average percentage of accepted tasks (%) 43
3.6 Effect of techniques on accepted tasks(%) 43

3.7 Average algorithm execution time(μs) 43
3.8 Effect of techniques on the algorithm execution time(%) . . . 43
3.9 Basic idea of quad-corner strategy 46
3.10 Examples of four task types and their alternative placement positions 47
3.11 The pseudocode of QC strategy 49
3.12 Evaluation with real hardware tasks 52
3.13 FPGA technology impact on penalty (%) and wasted area (%) 53

4.1 Performance parameters and previous algorithms 56
4.2 Our Intelligent Stuffing algorithm 58
4.3 Problem of scheduling and placement on 2D area model . . . 61
4.4 Basic idea of blocking-aware algorithm 62
4.5 3D total contiguous surface (3DTCS) heuristic 63
4.6 Horizontal contiguous surfaces 64
4.7 Vertical contiguous surfaces with scheduled tasks (a) and the FPGA boundary (b) 66
4.8 Pseudocode of 3D Compaction algorithm 68
4.9 Evaluation with synthetic workloads 71
4.10 Evaluation with real workloads 72
4.11 Evaluation of algorithm execution time 73

5.1 Logic elements 77
5.2 Basic operations of logic elements 78
5.3 Shortest clock timing of conventional (a) and our (b) logic elements 80
5.4 Simple circuit examples 82
5.5 Logic elements used in our experiments 84
5.6 A flip-flop circuit used in conventional LE experiments 85
5.7 A flip-flop circuit used in the proposed LE experiments 85
5.8 Power Reduction (%) 87
5.9 Area Overhead and Performance Improvement (%) 88

5.10 blif to conventional VHDL conversion (lion.blif to lion_conv.vhd) 90
5.11 lion benchmark example implemented using both coding styles 90
5.12 blif to our VHDL file conversion (lion.blif to lion_our.vhd) . . 91
5.13 Experimental setup 92
5.14 Overall power (%) reduction versus number of circuits (#Circuits) 95

6.1 Architecture of the conventional FPGA versus our proposed FPGA 101
6.2 Reconfiguration steps of the conventional FPGA versus our proposed FPGA 102
6.3 Mechanism of our proposed architecture 103
6.4 Hardware overhead for supporting fast reconfiguration 104
6.5 Bitstream for reconfiguration 105
6.6 Core relocation 106
6.7 Bitstream for core relocation 107

List of Acronyms and Symbols

3*DC*	3D Compaction
3*DTCS*	3D Total Contiguous Surface
ASICs	Application-Specific Integrated Circuits
AT	Arriving Task
BER	Balanced Empty Rectangles
BF	Best Fit
BL	Bottom-Left
BLIF	Berkeley Logic Interchange Format
BV	Boundary Value
CABAC	Context-Adaptive Binary Arithmetic Coding
CB	Clustering Based
CBP	Combine Before Placing
CLBs	Configurable Logic Blocks
CLCG	Combinational Logic Circuit Generator
CM	Configuration Memory
CPLDs	Complex PLDs
DC	Direct Combine
DFF	D Flip-Flop
DVS	Dynamic Voltage Scaling
EA	Earliest Available time
FAP	Fragmentation-Aware Placement
FH	Frame Height
FHC	Frame Height Counter
FHD	Frame Height Displacement
FHDR	Frame Height Displacement Register
FHR	Frame Height Register
FIR	Finite Impulse Response
FM	Fragmentation Matrix
FPD	Field-Programmable Device
FPGA	Field-Programmable Gate Array
FSM	Finite State Machine
GPP	General-Purpose Processor
HAP	Horizontal Alternative Position
HDL	Hardware Description Language
HL	Horizontal Line

HLS	High-Level Synthesis
ICAP	Internal Configuration Access Port
IM	Intelligent Merging
IPR	Impossible Placement Region
IS	Intelligent Stuffing
KAMER	Keeping All Maximal Empty Rectangles
KNER	Keeping Non-overlapping Empty Rectangles
LE	Logic Element
LER	Large Empty Rectangles
LIF	Least Interference Fit
LLT	Lower Left Task
LLTI	Lower Left Task Initial
LRT	Lower Right Task
LRTI	Lower Right Task Initial
LSEG	Longer Segment
LSQR	Large Square Empty Rectangles
LUT	LookUp Table
MCNC	Microelectronic Center of North Carolina
MER	Maximal Empty Rectangle
MKEs	Maximum Key Elements
MON	Merging Only if Needed
MR	Mask Register
NER	Non-overlapping Empty Rectangle
NPP	Nearest Possible Position
NSF	Next State Function
NST	Next Scheduled Task
OTF	On The-Fly
PLDs	Programmable Logic Devices
PM	Partial Merging
PPR	Possible Placement Region
PST	Previous Scheduled Task
QC	Quad-Corner
RCP	Routing-Conscious Placement
RPR	Reuse and Partial Reuse
SAF	Signal Activity File
SFTD	Sum of Finishing Time Difference
SHDR	Source Height Displacement Register
SL	Space List
SLA	Scan Line Algorithm

SPLDs	Simple PLDs
SQR	Square Empty Rectangles
SR	Shadow Register
SRAM	Static Random Access Memory
SSEG	Shorter Segment
SSFAR	Source Start Frame Address Register
ST	Scheduled Task
STM	Starting Time Matrix
SUR	Space Utilization Rate
TAAF	Time-Averaged Area Fragmentation
TAT	Total Algorithm Execution Time
TCM	Task Configuration Microcode
TFF	T Flip-Flop
THDR	Target Height Displacement Register
TL	Task List
TPA	Task Parameter Address
TPHRA	Task Placement Algorithm for Heterogeneous Reconfigurable Architectures
TRT	Total Response Time
TST	Total Schedule Time
TWA	Total Wasted Area
ULT	Upper Left Task
ULTI	Upper Left Task Initial
URT	Upper Right Task
URTI	Upper Right Task Initial
VAP	Vertical Alternative Position
VHDL	VHSIC Hardware Description Language
VHSIC	Very-High-Speed Integrated Circuit
VLS	Vertex List Set

1

Introduction

Nowadays, digital electronic systems are used in a growing number of real life applications. The most flexible and straight forward way to implement such a system is to use a processor that is programmable and can execute wide variety of applications. The hardware, a general-purpose processor (GPP) in this case, is usually fixed/hardwired, whereas the software ensures the computing system flexibility. Since such processors perform all workloads using the same fixed hardware, it is too complex to make the hardware design efficient for a wide range of applications. As a result, this approach cannot guarantee the best computational performance for all intended workloads. Designing hardware devices for a particular single application, referred as Application-Specific Integrated Circuits (ASICs), provides a system with the most efficient implementation for the given task, e.g., in terms of performance but often area and/or power. Since this requires time consuming and very costly design process along with expensive manufacturing processes, it is typically not feasible in both: economic costs and time-to-market. This solution, however, can become interesting when very high production volumes are targeted. Another option that allows highly flexible as well as relatively high performance computing systems is using reconfigurable devices, such as FPGAs. This approach aims at the design space between the full custom ASIC solution and the General-Purpose Processors. Often platforms of this type integrate reconfigurable fabric with general-purpose processors and sometimes dedicated hardwired blocks. Since such platforms can be used to build arbitrary hardware by changing the hardware configuration, they provide a flexible and at the same time relatively high performance solution by exploiting the inherent parallelism in hardware. Such systems where hardware can be changed at runtime are referred as runtime reconfigurable systems. A system involving partially reconfigurable (FPGA) device can change some parts during runtime without interrupting the overall system operation [1, 83, 154]. For example,

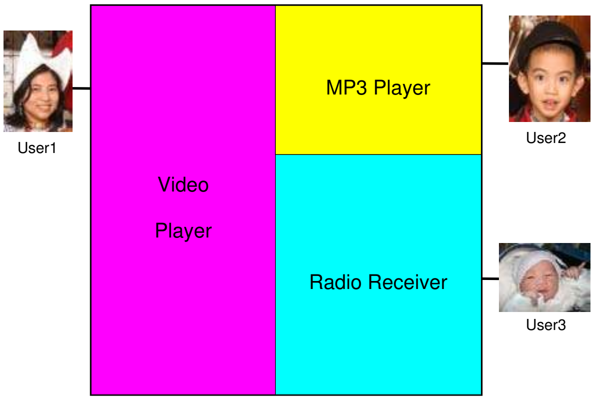


Figure 1.1: Running multiple applications concurrently on a partially reconfigurable device

while user1 is watching her exciting film, user2 can use part of the unoccupied hardware resources to enjoy listening to his favorite song without interrupting other user tasks running (e.g., the task used by user3 to listen to her favorite radio station) as illustrated in Figure 1.1. In this thesis, we target runtime reconfigurable systems that integrate tightly coupled general-purpose processor and a reconfigurable device, e.g., FPGA. This chapter provides the overview of the research presented in this dissertation. The main problems in runtime reconfigurable systems, addressed in this dissertation, are introduced in Section 1.1. In Section 1.2, we discuss briefly the main contributions of our work. The overall organization of this dissertation is presented in Section 1.3.

1.1 Problem Overview

Current devices used in runtime reconfigurable systems have the ability to reconfigure parts of their hardware resources without interrupting the normal operation of processing elements instantiated on the remaining fabric. Run time configuration has been used in several application areas and implementations, e.g., network crossbar switches [2], image interpolation [3], video coding [4], cryptography [5], neural network implementation [6], image processing [7], image compression [8], filters [9], matrix multiplications [9], motion estimation [10], mechatronics [11], Viterbi decoding [12] [14], multimedia player [13], Department of Defense (DOD) systems [15], Reed-Solomon

Coder/Decoder [16], among many others.

Exploiting partially reconfigurable devices for runtime reconfigurable systems can offer reduction in hardware area [6] [3] [4] [5] [8] [9] [11] [13], power consumption [5] [10] [12] [14] [61] [83] [84], economic cost [5], bitstream size [15], and reconfiguration time [15] [16] [83] [154] in addition to performance improvements [5] due to better resource customization. To make better use of these benefits, one important problem that needs to be addressed is hardware task scheduling and placement. Since hardware tasks are allocated and deallocated dynamically at runtime, the reconfigurable fabric can suffer of fragmentation. This can lead to undesirable situations where tasks cannot be allocated even if there would be sufficient free area available. As a result, the overall system performance will be penalized. Therefore, efficient hardware management of resources is very important.

Hardware task scheduling and placement algorithms can be divided into two main classes: *offline* and *online*. Offline assumes that all task properties (e.g., arrival times, task sizes, execution times, and reconfiguration times) are known in advance. The offline version can then perform various optimizations before the system is started. In general, the offline version has much longer time to optimize system performance compared to the online version. However, the offline approach is not applicable for systems where arriving tasks properties are unknown beforehand. In such general-purpose systems, the online version is the only possible alternative. In contrast to the offline approach, the online version needs to take decisions at runtime; as a result, the algorithm execution time contributes to the overall application latency. Therefore, the goal of the online scheduling and placement algorithms is not only to produce better scheduling and placement quality, they also have to minimize the runtime overhead. In this thesis, we focus on online scheduling and placement since we strongly believe it represents a more genetic situation. The online algorithms have to quickly find suitable hardware resources on the partially reconfigurable device for the arriving hardware tasks. In cases when there are no available resources for allocating the hardware task at its arrival time, the algorithms have to schedule the task for future execution.

Field-Programmable Devices (FPDs) are integrated circuits that can be (re)configured by their end users to implement various digital functions [17]. There are three FPD main categories: Simple Programmable Logic Devices (SPLDs), Complex PLDs (CPLDs) and Field-Programmable Gate Arrays (FPGAs) [17]. The main difference between PLDs and FPGAs is in the available number of logic inputs and the available logic capacities. While FPGAs have much higher logic capacity (and flip-flop to logic ratio), CPLDs offer more logic inputs. Advantages of using field-programmable devices (FPDs) in runtime reconfigurable systems are instant manufacturing turnaround, reduced start-up costs, low financial risk, short time-to-market and easy design changes [17]. However to get these benefits, the users need to pay additional costs: higher power consumption (approximately 12x larger dynamic power), larger silicon areas (40x more area required) and lower operating speeds (3.2x slower), as compared to the ASICs [18]. Higher power consumption requires expensive packaging [19] [20] [21], shortens chip life-times [19], asks for costly cooling systems [19] [20] [21], decreases system reliability [21] and prohibits battery operations [19] [20] [21]. Therefore, reducing the power consumption of FPDs (CPLDs and FPGAs) is a critical issue.

An FPGA device can be used to build arbitrary hardware circuits (same as any ASIC could implement) by reconfiguring and interconnecting its configurable logic elements (LEs) in different ways. Each LE contains a lookup table (LUT) as a configurable combinational circuit and a flip-flop (FF) as a storage element. A group of LEs forms configurable logic block (CLB). The CLB is the basic logic element used by Xilinx FPGAs. A somehow similar approach is used by Altera to organize LEs in clusters called logic array blocs (LABs). The complexity of the FPGAs in terms of available CLBs or LABs is growing very fast with the CMOS technology improvements and now allows the implementation of complete systems. Modern FPGAs can be used to implement circuits with complexity up to 474240 LUTs and 948480 FFs as shown in Table 1.1. The FPGA device can be reconfigured by changing the content of its configuration memory. The configuration memory content, called bitstream (BS) can be up to 185 Mbits and has to be transported to the FPGA internal memory using a dedicated configuration data path. Configuration data paths have usually limited bandwidth, hence, the time needed to send the configuration bits (called *reconfiguration time*) can be up to 58 ms for current technology with 32-bit wide configuration data path operating at 100 MHz. High reconfiguration time overhead can eclipse the benefits of dynamically reconfigurable systems. Therefore, it is very important to address this overhead. In addition, to use the FPGA resources more effectively and to cope with FPGA area fragmentation during runtime, one needs to easily reorganize the positions of running tasks, hence fast relocation is also necessary.

In this thesis, we assume pre-designed hardware tasks where for each task at least two options for execution exist: as software task on general-purpose processor or as hardware unit on the reconfigurable fabric. In the assumed system, each task can arrive at any time and its properties are unknown to the sys-

Table 1.1: Virtex FPGAs

FPGAs	LUTs	FFs	Bitstream size(Mbits)	Reconfiguration time(ms)
Virtex-4	178176	178176	51	16
Virtex-5	207360	207360	83	26
Virtex-6	474240	948480	185	58

tem beforehand. This models real situations in which the time when the user requests system resources for his/her usage is unknown. As a result, the envisioned system has to provide support for runtime hardware tasks placement and scheduling since this cannot be done statically at system design time. Similar to other work (e.g., [22]- [53]), we assume that each hardware task requires reconfigurable area with rectangular shape and can be placed at any location on the reconfigurable device. Our task model includes all required reconfigurable units and interconnect resources. Each hardware task is defined by three parameters: its *area* (width and height in terms of atomic reconfigurable units), *reconfiguration time*, and its *execution time* (the latter two expressed in system clock cycles).

The software tasks that are identified for hardware acceleration are first designed using a hardware description language and after that are placed and routed by commercial FPGA synthesis CAD tools to obtain functionally equivalent modules that can replace their respective software versions. At this step of the design process we can use the synthesis results to extract the task sizes for the used FPGA fabric. The output of the synthesis is the configuration bitstream that should be loaded to the device using its integrated configuration infrastructure. Therefore, starting from the task bitstream file, we can obtain precisely its reconfiguration time on the targeted technology. The two key ingredients are the configuration data size (the bitstream length in number of bits) and the throughput of the internal FPGA configuration circuit. As an example, the Internal Configuration Access Port (ICAP) of Virtex 4 FPGAs from Xilinx can transport 3200 million bits/second and will load a bitstream of size 51 Mbits in 15.9ms. The last parameter, the task execution time is specified by the time needed to process a unit of data (referred as: *Execution Time Per Unit of Data*, ETPUD) and the overall data size to be processed (i.e. how much data need to be processed). Please note that for some applications, the task execution time is also dependent on the exact data content (e.g., as in the case of Viterbi and Context-Adaptive Binary Arithmetic Coding (CABAC)). In such applications, even when processing the same amount of data, the elapsed time will be different when the input data content changes. To address data dependent task execution times, we envision two solutions: *worst case execution* *time* scenario and *notification on task completion* HW support.

In this thesis, we assume the worst case execution time scenario in which we use the task execution time when processing the worst case input data content. In such scenario, it is possible that the actual task completion can happen earlier than the scheduled completion time resulting in idle times that can not be used by the scheduler. In addition, such tasks will cause additional wasted area that cannot be utilized immediately by other HW tasks. In such non-optimal scenario, however, the overall computing system will operate correctly. Please note that the chosen scenario is the worst case in respect to the proposed placement and scheduling algorithms due to the introduced overheads in task execution time and wasted area. The second solution requires dedicated hardware support for feedback signaling when the running tasks complete, however, as mentioned earlier this can additionally improve the overall system performance. Some existing systems already have the necessary ingredients required to implement such support. For example, in the Molen architecture [203], the sequencer is aware of the hardware (HW) task start and completion timing. The only necessary extension in this case is to provide a way to pass this information to the HW scheduler and make it aware of running tasks completion. With this knowledge, the HW scheduler can make data content dependent tasks scheduling more efficient. This approach is outside of the scope of this thesis without loosing generality of our proposal. Even more the selected worst case execution scenario is less beneficial for the scheduling and placement algorithms presented in this thesis. We are strongly convinced that both types of systems will be able to benefit from the proposed algorithms.

The assumed overall system model used in our study is depicted in Figure 1.2 consisting of two main devices: the general-purpose processor (GPP) and the reconfigurable device (FPGA). All hardware task bitstream images are available in a repository resuming in the main memory (not explicitly shown on the figure) and can be requested by any running application on the GPP by using a dedicated operating system (OS) call. In response to such request, the OS will invoke the Placer (P) to find the best location on the FPGA fabric for the requested hardware task. Once appropriate location is found, the Translator will resolve the coordinates by transforming the internal, technology independent model representation to the corresponding low level commands specific for the used FPGA device. The Loader reads the task configuration bitstream from the repository and sends it to the internal configuration circuit, e.g., ICAP in case of Xilinx, to partially reconfigure the FPGA at the specific physical location provided by the Translator. After reconfiguring the FPGA fabric the requested hardware task execution is started immediately to avoid idle hardware units on

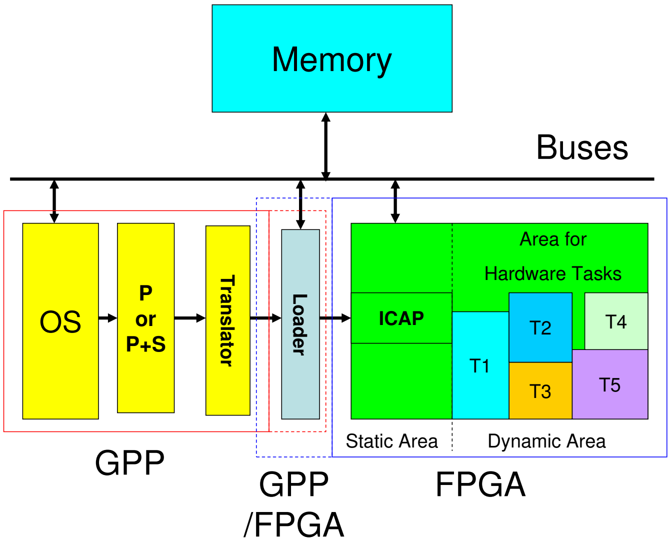


Figure 1.2: System Model (OS: operating system, P: placer, S: scheduler)

the reconfigurable fabric. For systems with combined Placer and Scheduler (P+S), when the hardware area is fully occupied, the Scheduler (S) schedules the task for future execution at predicted free area places at specific locations and specific starting times. For systems with only Placer (P), in cases when no hardware resources are available the corresponding task can be executed using its software version with significant and often unpredictable execution time penalty. The Loader can be implemented at either the general-purpose processor (GPP) as OS extension or in the Static Area of the FPGA. If the Loader is implemented in the GPP, the communication between the Loader to the ICAP is performed using the available off-chip connection. For implementations in the FPGA, the Loader connects to the ICAP internally. Similar to [202], dedicated buses are used for the interconnect on chip. Those buses are located at every row of the reconfigurable regions to allow data connections between tasks (or tasks and I/Os) regardless of the particular task sizes.

To illustrate the above processes at different stages of the system design and normal operation, we will use a simple example of hardware task creation as depicted in Figure 1.3. The hardware task in our example is a simple finite impulse response (FIR) filter. The task consumes input data from array A[i] and produces output data stored in B[i], where $B[i] = C0*A[i]+C1*A[i+1]+C2*A[i+2]+C3*A[i+3]+C4*A[i+4]$ and all data elements are 32 bits

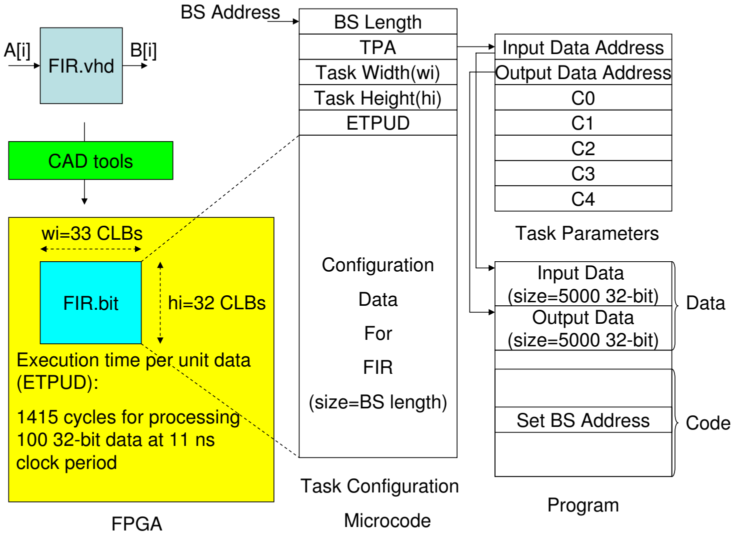

A[i]
FIR.vhd
B[i]
CAD tools
wi=33 CLBs
FIR.bit
hi=32 CLBs
Execution time per unit data (ETPUD):
1415 cycles for processing 100 32-bit data at 11 ns clock period
FPGA
BS Address
BS Length
TPA
Task Width(wi)
Task Height(hi)
ETPUD
Configuration Data For FIR (size=BS length)
Task Configuration Microcode
Input Data Address
Output Data Address
C0
C1
C2
C3
C4
Task Parameters
Input Data (size=5000 32-bit)
Output Data (size=5000 32-bit)
Data
Set BS Address
Code
Program

Figure 1.3: System at Design Time

wide. The task implementation described using a hardware description language (HDL) (FIR.vhd) is synthesized by commercial CAD tools that produce the partial bitstream file (FIR.bit) along with the additional synthesis results for that task. The bitstream contains the configuration data that should be loaded into the configuration memory to instantiate the task at a certain location on the FPGA fabric. The synthesis results are used to determine the rectangle area consumed by the task in terms of configurable logic blocks (CLBs) specified by the width and the height of the task. In our example, the FIR task width is 33 and the task height 32 CLBs for Xilinx Virtex-4 technology. Based on the synthesis output we determine the tasks reconfiguration times. Please note, that in a realistic scenario one additional design space exploration step can be added to steer task shapes toward an optimal point. At such stage, both, task sizes and reconfiguration times are predicted by using high-level models as the ones described in [54] in order to perform quick simulation of the different cases without the need of synthesizing all of the explored task variants. For example in Virtex-4 FPGA technology from Xilinx, there are 22 frames per column and each frame contains 1312 bits. Therefore one column uses $22x1312 = 28864$ bits. Since our FIR HW task requires 33 CLBs in 2 rows of 16 CLBs totaling in 32 CLBs, we obtain a bitstream with $33x2x28864 = 1905024$ bits. Virtex-4 ICAP can send 32-bit data every 100 MHz clock cycle, hence, we can estimate the reconfiguration time as $1905024x10/32 = 595320$ ns. Next,

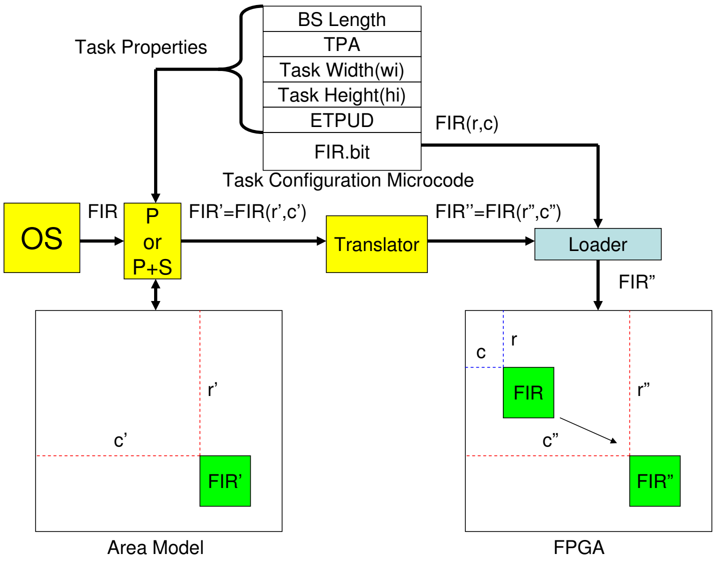


Figure 1.4: System at Runtime

the FIR task is tested by the designer to determine how fast the input data can be processed. For our example from Figure 1.3, the task needs 1415 cycles to process 100, 32-bit input data elements at 11 ns clock period making its ETPUD `1415x11 = 15565` ns per 100, 32-bit unit data. Based on the above ETPUD number, we can estimate the task execution time for various input data sizes. In our example, there are 5000, 32-bit input data elements that have to be processed by the FIR HW task. Therefore, the expected execution time of our FIR task is `(5000/100)x15565 = 778250` ns (778ms). The configuration data and task specific information are merged together in what we call a Task Configuration Microcode (TCM) block as shown in the upper part of Figure 1.3. TCM is pre-stored in memory at the Bitstream (BS) Address. The first field, the BS length represents the size of the configuration data field. This value is used by the Loader when the task is fetched from memory. The task parameter address (TPA) is needed to define where the task input/output parameters are located. In Figure 1.3, the task parameters are the input and output data locations, the number of data elements to be processed and the FIR filter coefficients (C0-C4). The input data address gives the location where the data to be processed remains. The location where the output data should be stored is defined by the output data address.

During runtime, when the system needs to execute the FIR hardware task on the FPGA, the OS invokes the Placer (P) to find a location for it (shown as example, in Figure 1.4). From the TCM the Placer gets the task properties information: task width, task height, reconfiguration time, and its execution time. Based on the current state of the used area model, the Placer searches for the best location for the task. When the location is found (in this example: r' and c'), the task is placed in the area model and its state is updated as shown on the left side of the figure. The Translator converts this location into real physical location on the targeted FPGA. In the bitstream file (FIR.bit), there is information about the FPGA location where the HW task was pre-designed (in this figure: r and c). By modifying this information at runtime, the Loader partially reconfigures the FPGA through the ICAP (by using the technology specific commands) at the location obtained from the Translator (r'' and c'' in this example). By decoupling our area model from the fine-grain details of the physical FPGA fabric, we propose an FPGA technology independent environment where different FPGA vendors, e.g., Xilinx, Altera, etc, can provide their consumers with full access to partial reconfigurable resources without exposing all of the proprietary details of the underlying bitstream formats. On the other side, reconfigurable system designers can now focus on partial reconfiguration algorithms without having to bother with all low-level details of a particular FPGA technology.

1.2 Research Questions and Main Contributions

As discussed in Section 1.1, efficient runtime reconfigurable systems management has to cope with the three main challenges: minimal hardware use, minimal power consumption, and minimal reconfiguration overhead. In respect to the above challenges the thesis at hand will provide an answer to the following research questions:

1. *How to improve area utilization, application execution time and algorithm overhead of hardware task placement and scheduling approaches?*
2. *How to reduce the power consumption of reconfigurable devices by improving their basic logic elements?*
3. *Can hardware task reconfiguration overhead be reduced by revisiting the configuration infrastructure?*

This dissertation elaborates on the above three questions critical for all modern runtime partially reconfigurable systems. The work contained in this dissertation provides evidence that the aforementioned questions can be successfully answered. More specifically, the main contributions of this thesis are:

1. Novel online hardware task scheduling and placement algorithms on partially reconfigurable devices with higher quality and shorter algorithm execution time compared to the state of the art;
2. A novel logic element for FPDs with reduced power consumption compared to the industrial FPDs;
3. Low overhead FPGA configuration circuit architectural extensions to shorten the reconfiguration and relocation times compared to current high-end FPGA devices.

1.3 Dissertation Organization

This dissertation consists of the following chapters:

- Chapter 2 gives an overview of the state of the art in solving problems in runtime reconfigurable systems. It presents a survey on existing online hardware task scheduling and placement algorithms, techniques to reduce power consumption in reconfigurable devices, and techniques to reduce reconfiguration overhead in runtime reconfigurable systems.
- Chapter 3 introduces two novel algorithms to deal with online hardware task placement problem in runtime reconfigurable systems on partially reconfigurable devices. The first algorithm (Intelligent Merging) deals with reducing algorithm execution time by avoiding unnecessary operations. The algorithm run up to 3x faster than related art with 0.89 % less placement quality. The second one (Quad-Corner) is more challenging, the aim is to discover a faster algorithm yet with a higher quality. The algorithm not only has higher placement quality (78 % less penalty and 93 % less wasted area) than related art but also has lower runtime overhead.
- Chapter 4 presents two novel online hardware task scheduling and placement algorithms on partially reconfigurable devices. The first algorithm (Intelligent Stuffing) is designed for 1D area model. The algorithm outperforms related art in terms of reduced total wasted area up to 89.7 %, has 1.5 % shorter schedule time and 31.3 % shorter waiting time. The

second one (3D Compaction) is proposed for 2D area model. The algorithm has up to 4.8 % shorter schedule time, 75.1 % shorter waiting time, and 22.9 % less wasted volume compared to related art.

- Chapter 5 describes a novel low power logic element (LE) for FPDs. FPDs using our proposal consume 6-90 % less total power and run 2-33 % faster than FPDs using conventional LEs.
- Chapter 6 shows a novel configuration circuit architecture for fast reconfiguration and relocation. The architecture can speedup reconfiguration and relocation by 4x and 19.8x on average, respectively.
- Chapter 7 summarizes our findings and gives some suggestions for future research directions.

2

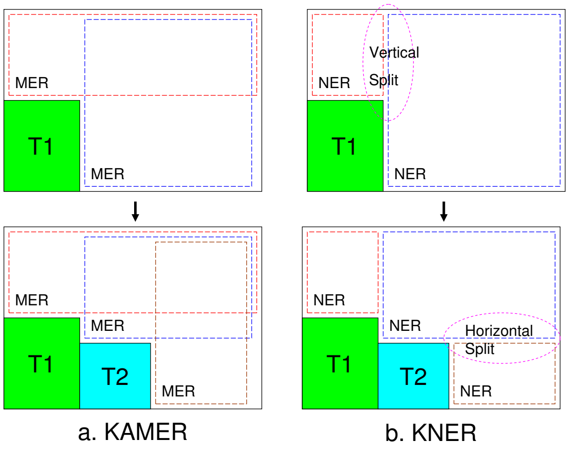
Related Work

As presented in Chapter 1, this dissertation targets three major problems in current runtime partially reconfigurable systems: online hardware task scheduling and placement, power consumption, and runtime reconfiguration overhead. Before presenting our proposal of how to address these problems in more detail, in this chapter, we present related work published by other researchers in their attempts to address the same problems.

This chapter is organized as follows. In Section 2.1, we discuss existing online task placement algorithms for partially reconfigurable devices. Related art regarding online task scheduling and placement is presented in Section 2.2. This section is short because work in this area is quite recent. In Section 2.3, we survey existing techniques for reducing power consumption in FPDs. Related work for reducing reconfiguration overhead is addressed in Section 2.4. Finally, we conclude this chapter in Section 2.5.

2.1 Online Hardware Task Placement Algorithms

In [22], Bazargan et al. proposed two algorithms: Keeping All Maximal Empty Rectangles (KAMER) and Keeping Non-overlapping Empty Rectangles (KNER). An empty rectangle is a rectangle area that can be used to place a task without overlapping to any running tasks on the FPGA. Such rectangle is used to place each arriving task at runtime. The two algorithms (KAMER and KNER) differ mainly in the way the empty rectangle is partitioned during task placements as shown in Figure 2.1. If the rectangle cannot be expanded anymore, the authors refer to it as a Maximal Empty Rectangle (MER); otherwise they call it a Non-overlapping Empty Rectangle (NER). KAMER organizes all MERs, whereas KNER manages all NERs. If there is an arriving task, both algorithms search all empty rectangles (i.e. MERs for KAMER or NERs for KNER) to find a suitable one which can fit the arriving task based on one of two-dimensional bin-packing heuristics: First Fit (FF), Best Fit (BF), and Bottom-left (BL). In KNER, only the selected empty rectangle is split into two smaller ones due to non-overlapping empty rectangles to decrease algorithm execution time of KAMER. The splitting can be done in one of two different ways: vertical split or horizontal split. For example, in Figure 2.1, the KNER decides to do vertical split after placing task T1 and horizontal split after placing task T2. Every time the algorithm places the task, it performs splitting operation. If it just splits the empty rectangles and does not merge them, FPGA will be partitioned into smaller and smaller of many empty rectangles. This situation will make placement quality worse, so the algorithm needs to do both merging and splitting operations. The split decision in KNER is made by utilizing one of these heuristics: Shorter Segment (SSEG), Longer Segment (LSEG), Square Empty Rectangles (SQR), Large Square Empty Rectangles (LSQR), Large Empty Rectangles (LER), and Balanced Empty Rectangles (BER). Although the authors have proposed these heuristics to avoid wrong splitting decisions, wrong decisions still cannot be totally avoided, lowering the placement quality of KNER. For example, in Figure 2.2, although the FPGA has enough free area for a new incoming task, KNER rejects it due to its incorrect splitting decision in the past.



a. KAMER b. KNER

Figure 2.1: KAMER vs KNER

In [23] and [24], Tabero et al. proposed a Vertex-list algorithm. Vertex List Set (VLS) data structure is used that each of the lists describes the contour of each unoccupied area fragment in the FPGA. The authors use bottom-left or top-right heuristic for placing arriving tasks on vertexes. The VLS structure is a

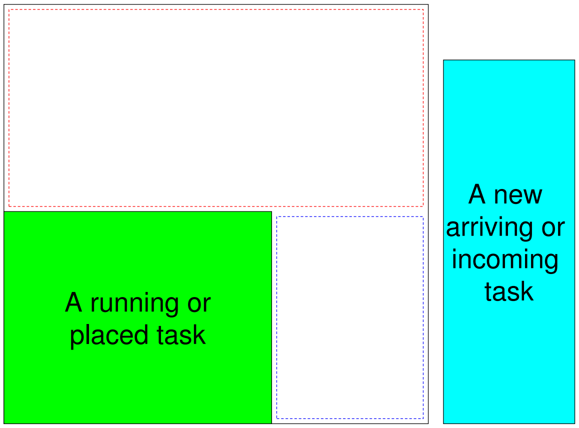


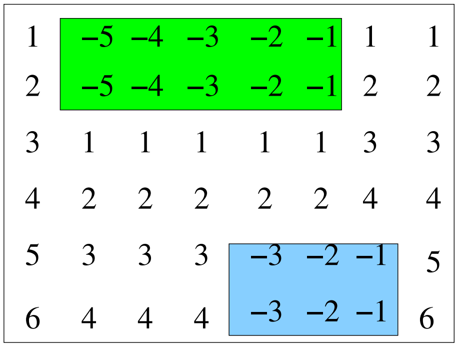
Figure 2.2: A rejection of an arriving task due to its previous wrong splitting decision

geometrical description of the whole FPGA free area perimeter. Instead of partitioning free areas as KAMER and KNER, the Vertex-list algorithm focuses on the free area perimeter for placing an arriving task. The algorithm places a task on one of the corners of this free area perimeter based on one of two heuristics: 2D-adjacency-based and fragmentation-based. The 2D-adjacency-based heuristic inserts the task on the vertex position that has the highest contact level in space of the task to the running tasks and the FPGA boundary, while the fragmentation-based heuristic inserts the task on the vertex position that has the lowest fragmentation level. To enhance placement quality of their Vertex-list algorithm, in [25], they proposed two new heuristics: 3D-adjacency and Look-ahead 3D. In addition to the contact level in space, the 3D-adjacency heuristic also computes the contact level in time to pack tasks in space and time. In Look-ahead 3D heuristic, the 3D-adjacency value is computed not only at current time but also at the future time when the next running task is finished. After computing all the 3D-adjacency values, the task is placed at the position with the highest 3D-adjacency value.

In [26] and [27], Steiger et al. and Walder et al. proposed an On The-Fly (OTF) algorithm. As mentioned earlier, the wrong split decision can lower the placement quality of KNER. To avoid such wrong decisions, they modified KNER by delaying the split decision until a next arrived task placed on the FPGA. However, for this modification, they need to resize several rectangles on a task insertion.

In [28], Morandi et al. proposed a Routing Aware Linear Placer (RALP) algorithm. The algorithm is a modified version of KNER algorithm with an additional routing cost consideration between dependent tasks. Tasks are placed on empty rectangles that have the least Manhattan distances between dependent tasks to minimize routing cost. The algorithm can reduce routing cost compared to KNER as reported in [28]. However, it has a lower placement quality that KNER due to the negative impact of its routing consideration.

In [29], Ahmadinia et al. proposed a Horizontal Line (HL) algorithm to manage free space and the Clustering Based (CB) strategy to improve the placement quality of their HL approach. Instead of managing a list of empty rectangles like in KAMER, KNER [22], and OTF [26] [27], HL uses exactly two horizontal lines for placing the task; one above (top horizontal line) and one below (bottom horizontal line) the placed tasks. In order to store information on these horizontal lines, HL uses two separate linked-lists. HL is implemented in such a way that hardware tasks are placed above the currently running tasks as long as there is free space. Once there is no empty space found above the running tasks, the new ones start to be placed below them and so on. The basic idea of CB is to place all tasks with similar end times in the same strip such that a large empty space will be created at a certain location at the end time. This new empty space hopefully will be able to accommodate future larger tasks.



1	−5	−4	−3	−2	−1	1	1
2	−5	−4	−3	−2	−1	2	2
3	1	1	1	1	1	3	3
4	2	2	2	2	2	4	4
5	3	3	3	−3	−2	−1	5
6	4	4	4	−3	−2	−1	6

Figure 2.3: Area matrix of the staircase algorithm

In [30] and [31], Handa et al. proposed the Staircase algorithm for finding MERs. A 2D array, referred as area matrix, for modeling the FPGA surface with each cell in the array representing a CLB is used. For example, an FPGA (8x6 CLBs) with two running tasks (5x2 CLBs and 3x2 CLBs) is modeled with an area matrix in Figure 2.3. Every CLB array cell in the free area contains a positive number that gives the number of continuous empty cells in the column above including the cell itself. Every cell in the occupied area holds a negative number that represents the remaining width of the task measured on

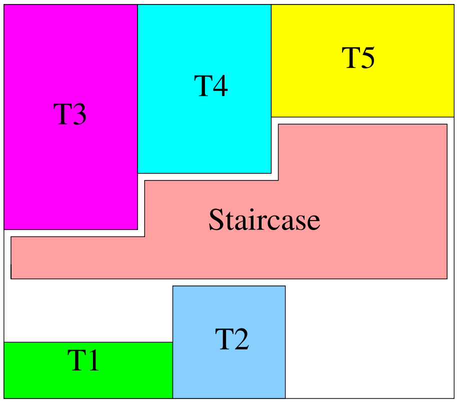
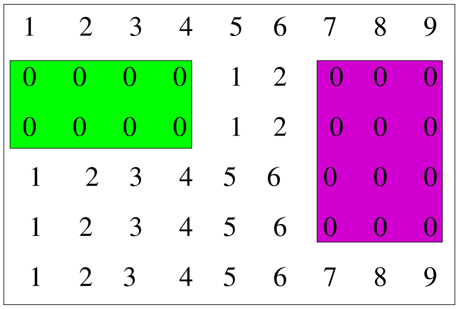


Figure 2.4: A staircase

the right side from that cell. The area matrix is used for constructing staircases and finally these staircases are utilized for finding MERs. All the empty rectangles having same bottom right coordinate make a staircase as illustrated in Figure 2.4. They only check maximal staircases for extracting MERs. After a maximal staircase is detected, all the rectangles in the staircase can be checked to see if they are MERs.

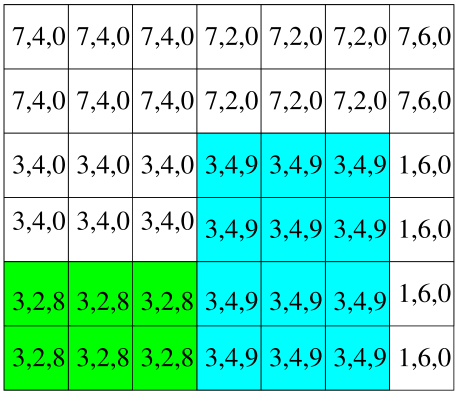


1	2	3	4	5	6	7	8	9
0	0	0	0	1	2	0	0	0
0	0	0	0	1	2	0	0	0
1	2	3	4	5	6	0	0	0
1	2	3	4	5	6	0	0	0
1	2	3	4	5	6	7	8	9

Figure 2.5: Area matrix of SLA

In [32], Cui et al. proposed the Scan Line Algorithm (SLA) algorithm for finding MERs. The authors use the same area matrix as the staircase algorithm with a different encoding to represent the FPGA area. For example, an FPGA (9x6 CLBs) with two running tasks (4x2 CLBs and 3x4 CLBs) is modeled with an area matrix in Figure 2.5. Every free area CLB is represented by a positive number that gives the number of continuous empty cells on left including that cell as shown in Figure 2.5. Every occupied CLB is represented by a zero. In SLA, the area matrix is used for finding Maximum Key Elements (MKEs) and finally these MKEs are scanned for finding all MERs. A key element is an empty cell with an occupied cell as its right hand neighbor. A column that contains one or more key elements is called as a scan line. A scan line is partitioned into segments by valley points. A valley point is the element within a scan line where values in the line starting to increase. Each segment has one MKE which is the largest key element in the corresponding segment.

In [33], Xiao et al. discovered that the SLA algorithm can find empty rectangles that are not MERs. To solve this problem, they proposed ESLA algorithm in [33]. Before the algorithm scans each MKE for finding MERs, the algorithm is instrumented with an ability to know a set of valid widths for that corresponding MKE. By doing so, the algorithm only needs to scan an MKE for those valid widths. To avoid duplicated scanning, the algorithm records a set of scanned positions during each MKE scanning.



7,4,0	7,4,0	7,4,0	7,2,0	7,2,0	7,2,0	7,6,0
7,4,0	7,4,0	7,4,0	7,2,0	7,2,0	7,2,0	7,6,0
3,4,0	3,4,0	3,4,0	3,4,9	3,4,9	3,4,9	1,6,0
3,4,0	3,4,0	3,4,0	3,4,9	3,4,9	3,4,9	1,6,0
3,2,8	3,2,8	3,2,8	3,4,9	3,4,9	3,4,9	1,6,0
3,2,8	3,2,8	3,2,8	3,4,9	3,4,9	3,4,9	1,6,0

Figure 2.6: Fragmentation matrix of CF

In [34], Cui et al. proposed the Cell Fragmentation (CF) algorithm for their online placement algorithm. CF uses the SLA to find MERs and a Fragmentation Matrix (FM) to represent the area of the FPGA. For example, an FPGA (7x6 CLBs) with two running tasks (3x2 CLBs and 3x4 CLBs) is modeled with a fragmentation matrix in Figure 2.6. For empty cells, each cell is labeled by the number of contiguous empty cells in horizontal, in vertical direction, and a zero. For occupied cells, each cell is labeled by the number of contiguous occupied cells in horizontal, in vertical direction, and the finish time of the task. In order to place a task on the FPGA, CF calculates the Time-Averaged Area Fragmentation (TAAF) for all MERs that are large enough to accommodate the task and then places the task into one of the MERs which has the largest TAAF. The TAAF of a MER is the degree of impact of this MER on the overall degree of fragmentation. A MER with large TAAF means it has more impact on the overall degree of fragmentation, that's why CF places the arriving task to a MER that has the largest TAAF.

In [35] and [36], Tomono et al. proposed an online FPGA algorithm that does not only take into consideration the degree of fragmentation, but also the speed of I/O communication computed based on the Manhattan distances. The aim of their algorithm is to balance the degree of fragmentation and the speed of I/O communication. They use the same area matrix data structure as used by the Staircase algorithm with additional I/O communication constraints, so they increase the degree of fragmentation in order to gain the speed of I/O communication. Because of this additional consideration, they need to check the status of each communication channel during staircase creation.

In [37] and [38], Ahmadinia et al. proposed the Nearest Possible Position (NPP) algorithm. They manage the occupied space rather than the free space, because the set of empty rectangles grows much faster than the set of placed rectangles. The impossible placement region (IPR) of an arriving task relative to a placed task is the region near the placed task where it is impossible to place this arriving task without overlapping the placed task. The possible placement region (PPR) is the area where it is possible to place the arriving task without overlapping any placed tasks. In order to find the best position on the PPR for placing an arriving task, they compute the routing cost based on Euclidean distances and place an arriving task at the optimal point, where routing cost is minimum. If they cannot find the optimal point on the PPR, they will find the Nearest Possible Position (NPP) for placing the task.

In [39], Lu et al. proposed the Multi-Objective Hardware Placement (MOHP) algorithm. The algorithm uses the VLS data structure adopted from [23] and [24]. Incoming tasks are classified into three groups with different treatments. The first group is for independent tasks that need to be executed urgently due to short remaining time to the deadline. To handle tasks in this group, the algorithm uses the FF heuristic for fast allocation. The second group is for independent tasks that do not need urgent treatment. For this group, the algorithm adopts a vertex-list approach from [23] and [24]. The third group is for dependent tasks. In this group, the routing between dependent tasks needs to be shortened. For that reason, the algorithm utilizes the NPP approach adopted from [37] and [38] for tasks in this group.

In [40], Elbidweihy and Trahan proposed an online placement algorithm that manages both Maximal Horizontal Strips (MHS) and Maximal Vertical Strips, called the Maximal Horizontal and Vertical Strips (MHVS) algorithm. MHSs are rectangles generated by partitioning free area using top and bottom boundaries of running tasks; whereas MVSs are rectangles that result from free area partitioning using left and right boundaries of running tasks. In this algorithm, the first fit rectangle is used for placing an arriving task. The algorithm can run faster compared to KAMER as reported in [40]. However, it has a lower placement quality than KAMER.

In [41] and [42], Ahmadinia et al. proposed the Routing-Conscious Placement(RCP). In order to reduce free-space management to a single point, they expand inserted modules and concurrently shrink the FPGA area and an arriving task by half both in width and height. To choose the position for placing an arriving task, they choose a position, such that the weighted communication cost computed based on the Manhattan distances is minimized.

In [43], Köster et al. proposed a task placement algorithm for heterogeneous reconfigurable architectures, TPHRA. The basic idea of this algorithm is to avoid placing an arriving task with many feasible positions in areas that can be used by tasks with few feasible positions whenever possible.

In [44], Ahmadinia and Teich proposed the Least Interference Fit (LIF) algorithm. In order to reduce the reconfigurable overhead due to the limitation of currently available FPGA technology in that time (column-wise reconfigurable capability), LIF places tasks at the position where the tasks interfere with the currently running tasks as little as possible.

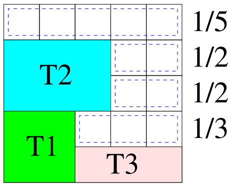


Figure 2.7: FAP

In [45], ElFarag et al. proposed the Fragmentation-Aware Placement (FAP) algorithm. In this paper, they introduced a fragmentation metric that gives an indication to the continuity of occupied (or free) space on the reconfigurable device and not the amount of occupied (or free) space. The algorithm places each arriving task on the location where the fragmentation metric is smallest. All empty spaces have to be tested before it can select one that causes the lowest fragmentation. Figure 2.7 shows how they compute the fragmentation metric. There are three tasks (T1, T2, and T3) placed on a reconfigurable device with size of 5x5 reconfigurable units. In the first row of reconfigurable device, there is one contiguous empty space that consists of five reconfigurable units, therefore the fragmentation in this row is 1/5. Using similar way, the total row fragmentation is `((1/5) + (1/2) + (1/2) + (1/3))`, while the total column fragmentation is `((1/1) + (1/1) + (1/1) + (1/1) + (1/4) + (1/4))`.

2.2 Online Task Scheduling and Placement

In [46] and [47], Steiger et al. proposed the Horizon and Stuffing algorithms both for 1D and 2D area models. The Horizon guarantees that arriving tasks are only scheduled when they do not overlap in time or space with other scheduled tasks. The Stuffing schedules arriving tasks to arbitrary free areas that will exist in the future by imitating future task terminations and starts. In these papers, the authors reported that the Stuffing algorithm outperforms the Horizon algorithm in scheduling and placement quality.

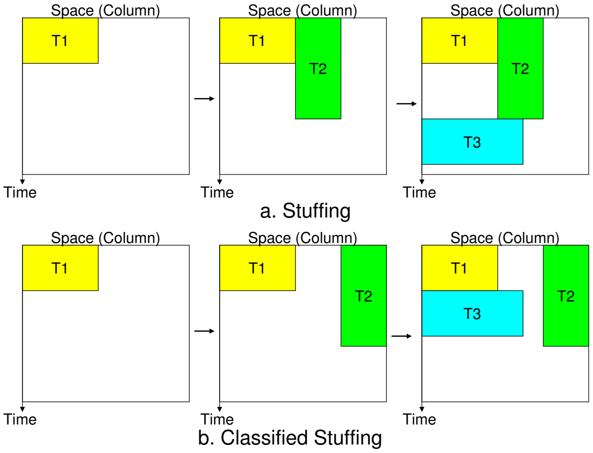


Figure 2.8: Stuffing vs Classified Stuffing

Discovered that the problem of the Stuffing that always places a task on the leftmost of its free space as shown in Figure 2.8a, Chen and Hsiung in [48] proposed their 1D Classified Stuffing. By classifying incoming tasks before scheduling and placement, the 1D Classified Stuffing performs better than the original 1D Stuffing. For example, because the Stuffing algorithm always places tasks on the leftmost edge of the available area, it places tasks T1 and T2 as shown in Figure 2.8a. These placements block task T3 to be scheduled earlier. In this case, the Stuffing fails to place task T3 earlier. The main difference between the Classified Stuffing and the Stuffing is the classification of tasks. The Classified Stuffing can place a task on the leftmost or rightmost of its free space based on the task Space Utilization Rate (SUR). SUR is the ratio between the number of columns required by the task and its execution time. High SUR tasks (SUR > 1) are placed starting from the leftmost available columns of the FPGA space, while low SUR tasks (SUR ≤ 1) are placed from the rightmost available columns. For this simple example, the Classified Stuffing can recognize the difference between tasks T1 (high SUR task) and T2 (low SUR task), so it places successfully tasks on different sides as shown in Figure 2.8b. Therefore the task T3 can be scheduled earlier by the Classified Stuffing, outperforming the Stuffing as reported in [48].

In [49], Marconi et al. proposed their 1D Intelligent Stuffing to solve the problems of both the 1D Stuffing and Classified Stuffing. The main difference between their algorithm and the previous 1D algorithms is the additional alignment flag of each free segment. The flag determines the placement location of the task within the corresponding free segment. By utilizing this flag, the 1D Intelligent Stuffing outperforms the previously mentioned 1D algorithms.

In [50], Lu et al. introduced their 1D reuse and partial reuse (RPR). The algorithm reuses already placed tasks to reduce reconfiguration time. As a result, the RPR outperforms the 1D Stuffing.

In [51], Zhou et al. proposed their 2D Window-based Stuffing to tackle the drawback of 2D Stuffing. By using time windows instead of the time events, the 2D Window-based Stuffing outperforms previous 2D Stuffing. The drawback of their 2D Window-based Stuffing is its long execution time. To reduce this runtime cost the authors proposed the Compact Reservation (CR) in [52]. The main idea of the CR is the computation of the earliest available time (EA) matrix for every incoming task. That contains the earliest starting times for scheduling and placing the arriving task. The CR outperforms the original 2D Stuffing and their previous 2D Window-based Stuffing.

2.3 Low Power Techniques for Reconfigurable Devices

Modern FPGAs contain embedded hardware blocks, such as: multipliers, DSPs, and memories. It is reported in [18] [61] that mapping designs to these blocks can reduce power consumption. The design that uses hard blocks requires less interconnection. As a result, static and dynamic power consumptions are reduced.

Adding programmable delay circuits into configurable logic blocks is reported in [68] to reduce power consumption in FPGAs. The generation of glitches is avoided by aligning the arrival times of signals using the proposed programmable delay circuits. As a result, the glitched are reduced for minimizing dynamic power consumption.

To reduce dynamic power consumption in FPGAs, circuits are pipelined in [64] [65] [66]. This technique reduces the number of levels of the circuit between registers by dividing the circuit into stages. A circuit with lower levels tends to produce fewer glitches. Since a circuit with fewer glitches consumes less dynamic power, the power consumption is reduced.

Reducing power consumption in FPGAs by inserting negative edge triggered flip-flops at the outputs of selected LUTs to block glitches for propagating further is reported in [71]. Since the technique produces a circuit with fewer gliches, the dynamic power consumption is reduced.

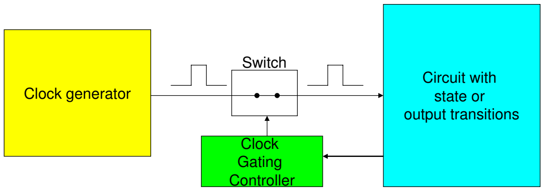
Retiming can be used to reduce dynamic power consumption in FPGAs [67]. The idea is to redistribute registers along a signal path without changing the functionality of the circuit. By doing so, the logic between registers is minimized, hence reducing glitches. As a result, the dynamic power consumption is reduced.

The bit-widths of the internal signals of circuits can be optimized to reduce dynamic power consumption. A circuit with shorter bit-widths consumes less power. This approach applied in FPGAs is reported in [72] [73].

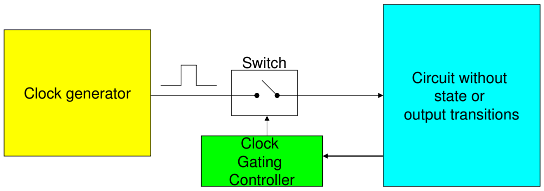
Clock gating is used to reduce dynamic power consumption by selectively blocking the circuit local clock when no state or output transition takes place as illustrated in Figure 2.9. The clock gating controller is needed for detecting the conditions of the observed circuit. Based on these conditions, the clock gating controller can know the exact time when it can stop clock signal to be transported to the specific circuit for power saving. It is used in FPGAs [74] [75] [76] [77] [61] [78] [79] [80] and CPLDs [81]. This technique is supported by commercial CAD tools from Xilinx as reported in [77]. In [80], an asynchronous FPGA with clock gating is proposed.

Powering FPGAs with a variable supply voltage can also be used to reduce power consumption [82]. This method is referred as dynamic voltage scaling (DVS). Since there is a quadratic relationship between supply voltage and dynamic power, reducing the voltage will significantly reduce the dynamic power. Moreover, a cubic relationship between supply voltage and leakage power reduces significantly the leakage power.

Modern FPGAs have the ability to reconfigure part of their resources with-



a. Transporting clock signal for normal operation



b. Stop transporting clock signal for power saving

Figure 2.9: Clock gating

out interrupting the remaining resources at runtime. Hardware sharing can be realized by utilizing this partial reconfiguration feature for power consumption reduction. Power saving using this approach in FPGAs is reported in [5] [10] [12] [14] [61] [83] [84].

Clock scaling is an approach to reduce power consumption by adjusting operating clock frequency dynamically. Applying this approach in FPGAs is reported in [85].

A lower threshold voltage transistor runs faster, but it consumes more power. Multi-threshold voltage technique is to use higher threshold voltage transistors on noncritical paths to reduce static power, and low threshold voltage transistors on critical paths to maintain performance. This technique has been applied in commercial FPGAs as reported in [61] [86].

A lower capacitive circuit consumes less dynamic power. One of the ways to reduce capacitance is to use a low-k dielectric material. This technique is used by commercial FPGAs as shown in [61] [86].

A simple way for static and dynamic power savings is to reduce the supply voltage. Commercial FPGAs reported this in [61] [86].

Building circuits with bigger size lookup tables (LUTs) needs less interconnection between LUTs. As a result, interconnect power consumption is reduced. This has triggered commercial FPGA vendors to use bigger size LUTs as reported in [61] [63].

Power gating is a technique for reducing power consumption by temporarily turning off circuits that are not in use. It is applied in FPGAs [87] [88] [89] [90] [91]. This technique is used in commercial products, such as: Atmel PLDs [93], Altera CPLDs [94], Actel FPGAs [95], QuickLogic FPGAs [96], Xilinx FPGAs [61] , Altera FPGAs [92]. In [88], an asynchronous FPGA with autonomous fine-grain power gating is proposed. How to partition a design to better benefit from power gating technique is reported in [91].

Conventional single-edge-triggered flip-flops respond only once per clock pulse cycle. To reduce power consumption, a flip-flop that can respond to both the positive and the negative edge of the clock pulse (double-edge-triggered flip-flops) was proposed in [97]. This technique is used in Xilinx CPLDs to reduce power consumption [98].

Since SRAM memory is volatile, SRAM-based FPGAs need to be reconfigured before usage. This reconfiguration consumes power. In contrast, the flash-based FPGAs (e.g. Actel FPGA [95]) that use non-volatile memory can be operated directly without reconfiguration.

Powering FPGAs with two different supply voltages (dual-Vdd) can also reduce power consumption as reported in [20] [99] [100] [101] [102] [103]. It is to use lower supply voltages on noncritical paths to reduce power, and higher supply voltages on critical paths to maintain performance. Algorithms for Vdd assignment are presented in [101] [102]. [103] combines concurrently this technique with retiming to better reduce power consumption in FPGAs.

Reordering input signals to LUTs can reduce dynamic power consumption in FPGAs. By doing so, we can minimize the switching activity inside LUTs as reported in [104]. Since power consumption depends linearly on the switching activity, reducing this results in power consumption improvement.

Power consumption in FPGAs can be reduced by dividing a finite state machine (FSM) into two smaller sub-FSM using a probabilistic criterion [105]. The idea is to activate only one sub-FSM at a time, meanwhile the other is disabled for power reduction. Choosing state encoding of FSM for power reduction in FPGAs is reported in [106] [107]. The idea is to minimize the bit changes during state transitions for reducing switching activity, hence minimizing the dynamic power consumption.

Using a diagonally symmetric interconnect pattern in Virtex-5 FPGAs can reduce the number of interconnect routing hops as reported in [108]. As a result, the interconnect power consumption is reduced.

Since not all inputs of LUTs are used in real FPGA designs, leakage power can be reduced by shutting off SRAM cells and transistors associated with unused

LUT inputs as reported in [21].

Using LUTs able to operate in two different modes (high-performance and low-power) reduces leakage power as reported in [109]. The idea is to use some transistors for lowering supply voltage across input inverters of LUTs during low power operation mode. Since not all LUTs need to be operated in high-performance mode, the leakage power is reduced.

Resources used by tasks cannot be turned off after configuration, consuming leakage power. Therefore, tasks need to be operated as soon as possible after configuration in runtime systems using partially reconfigurable devices. This technique for leakage power reduction in FPGAs is reported in [110].

Since leakage power in multiplexers is dependent on their input states, selecting polarities for logic signals (i.e. inverted or not) so that the multiplexers are operated in low-leakage states in the majority of time can be used to reduce leakage power in FPGAs [111]. To reduce more leakage power, the work in [111] is extended by [112]. In [112], not only polarity is considered to achieve low leakage states, but also the order of input signals to LUTs is modified to have a better leakage power reduction. It is different from [104] that targets dynamic power, the work in [112] targets static power by reordering input signals to LUTs. Since the leakage power is state dependent [113], changing this state results leakage power reduction.

Redesigning routing switches can reduce the FPGA overall power consumption. Routing switches that can operate in three different modes: high-speed, low-power or sleep is reported in [114]. Using dual-Vdd-dual-Vt routing switches for reducing interconnect power is presented in [115]. Applying dual-vdd and power gating techniques for routing switches is proposed in [116].

During high-level synthesis(HLS), a circuit can be implemented by combining functional units, such as: multipliers, adders, multiplexers, etc. Each functional unit can be realized using one of varied implementations. Each implementation requires a certain area and runs at a specific speed with required power consumption. To reduce power consumption, we need to choose the best design for a given circuit that can meet the timing requirement with minimal power. HLS algorithms for minimizing power consumption in FPGAs are reported in [117] [118].

Logic synthesis in FPGAs is a process of transforming a given design (coded in schematic or HDL) into a gate-level circuit. Considering switching activity during logic synthesis for FPGAs to reduce power consumption is presented in [119]. The idea is to minimize switching activity during logic synthesis. As a result, the power consumption is reduced.

Technology mapping in FPGAs is a process of transforming a given circuit into a circuit that only consists of LUTs. The way we map circuits into FPGAs can affect the power consumption. The algorithms to perform this process for power reduction are presented in [120] [121] [122] [123] [124]. The main idea is to pack nodes with high switching activity inside LUTs. By doing so, we can minimize power needed to transport signals of nodes among LUTs. To better estimate the switching activity, glitches are considered during technology mapping in [70].

Transformation by changing the functionalities of LUTs with rerouting [125] and without rerouting [126] can be used to reduce power consumption in FPGAs. [125] performs the transformation after technology mapping by reducing switching densities of the outputs of the LUTs, whereas [126] transforms the design after mapping, placement, and routing by considering switching activity and capacitance at the outputs of the LUTs.

Clustering logic blocks in FPGAs can affect reduction in power consumption. Clustering reduces the usage of interconnect resources. As a result, it reduces interconnect power. The optimal number of logic elements per cluster for power reduction is 12 as reported in [127]. The way we cluster a circuit into an FPGA can affect the power consumption. The clustering algorithms to reduce power consumption are presented in [128] [129]. The main idea in [128] is to minimize intercluster connections for reducing interconnection power. Clustering for FPGAs with dual-Vdd is shown in [129]. Assigning noncritical paths to clusters with low power supply voltage is the key idea of [129].

Placement algorithms to reduce power consumption in FPGAs are presented in [130] [131] [132]. The main idea is to add estimated dynamic power into cost function of the placement algorithms. As a result, dynamic power is reduced during placement. A placement algorithm that takes into account the cost of using clock network resources to reduce power consumed by clock network is reported in [132].

Routing algorithms to reduce power consumption in FPGAs are reported in [130] [69]. Assigning nodes with high switching activity to low-capacitance routing resources is the main idea behind the routing algorithm for reducing interconnect power in [130]. A routing that can balance arrival times of the inputs of the same LUTs to reduce power consumption in FPGAs is proposed in [69]. By doing so, the glitches are reduced. As a result, the dynamic power consumption is minimized.

Combining power-aware technology-mapping, placement, and routing algorithms to reduce FPGA power consumption is reported in [133].

To reduce power during runtime reconfiguration, configuration memory with two different types of memories [134] or runtime configurable memory with two different modes [135] is proposed. One type(mode) is optimized for high speed operation; whereas the other type(mode) is optimized for low power operation. Tasks that do not require high speed reconfiguration can be reconfigured to the low power one for power saving during reconfiguration.

Some signals in a digital circuit do not affect an output of the circuit for certain conditions. Stopping these signals to flow to the circuit at those conditions for dynamic power saving in FPGAs is reported in [136] [137].

Choosing the best operating mode for each memory on FPGAs based on prior knowledge of its dead intervals is reported in [138] to reduce leakage power consumption. The memory can be operated in three operating modes: active, drowsy, and sleep. The sleep mode is a condition when the power supply is disconnected to the memory; whereas the drowsy mode is a condition when the memory is connected to a lower supply voltage. The idea is to operate the memory based on its dead intervals. The memory with long/medium/short dead interval is operated on sleep/drowsy/active mode.

Constraining designs to be implemented on the specific regions within the FPGA to minimize power consumed by clock networks is reported in [139]. The idea is to place logic closer together for minimizing the clock network usage. As a result, the FPGA power consumption is reduced.

Using nanoelectromechanical relays for programmable routing in FPGAs is reported in [140] to reduce power consumption due to its zero leakage and low on-resistance characteristics. Although it is more power efficient than the conventional FPGA, it is not suitable for runtime reconfigurable systems due to its large mechanical switching delay.

Older generation FPGAs use dual-oxide process technology: thick oxide transistors (slow transistors) for I/Os and thin oxide transistors (fast transistors) for core. To reduce leakage power in FPGAs, triple-oxide process technology is used in modern FPGAs (e.g. Virtex-4) [62] [61]. In these FPGAs, another type of transistors with medium thickness oxide is dedicated for the configuration memory and interconnect pass gates.

The leakage power consumed by an asymmetric SRAM cell depends on its stored data. Since 87 % of the configuration memory cells in FPGAs store logic zero in the real FPGA design [141], using asymmetric SRAM cells with low leakage at logic zero for FPGAs to reduce leakage power consumed by reconfiguration memory is reported in [141]. The idea is to select polarities for logic signals (i.e. inverted or not) that can increase the number of zeros stored on the configuration memory. Since the number of zeros is increased, the number of memory cell that operates at low leakage is increased. As a result, the leakage power consumed by the reconfiguration memory is reduced.

To reduce interconnect power, low-voltage swing interconnects are applied for FPGAs in [142] [143]. Since the dynamic power consumption is linearly proportional to the voltage swing, interconnect power is reduced by minimizing the voltage swing on interconnects. Because this technique degrades the performance, in [142], the dual-edge triggered flip-flops are used to handle this degradation. Applying low swing interconnects only on non-critical paths is proposed in [143] to reduce the performance degradation of this technique.

2.4 Reconfiguration Overhead Reduction Techniques

One of the ways to alleviate the reconfiguration time penalty is to widen the FPGA configuration data path as shown in the Xilinx FPGA evolution from Virtex-II (8-bit configuration data path [156]) to Virtex-4 (32-bit configuration data path [56]). Since the data path is wider, more configuration data can be sent in each clock cycle. As a result, the configuration time is reduced.

Partially reconfigurable FPGAs can also be used to shorten the reconfiguration times [1] [83] [15] [16] [154]. In this case, we do not need to reconfigure the whole fabric when we want to change the systems, only part of systems that needs to be changed is required for reconfiguration. The architecture of partially reconfigurable FPGAs is shown in Figure 2.10a. The operation of this FPGA is illustrated in Figure 2.10b. A frame of configuration data is loaded serially into a shift register (Configuration Register (CR)) at times t=t1 to t=t5 as illustrated in Figure 2.10b. After the entire frame is loaded into CR, it is temporarily transferred to a Shadow Register (SR) (Figure 2.10b at t=t6) so that the CR is free to begin receiving the next frame of data. An address line is used to transfer the data from the shadow register via the data lines into the selected Configuration Memory (CM) cells as illustrated in Figure 2.10b at t=t7. The Mask Register (MR) selects which memory cells receive the specific configuration data values and which memory cells do not, thereby defining a partial reconfiguration zone as shown in the same figure. As a consequence, the configuration time is shortened.

To reduce reconfiguration overhead, configuration prefetching is proposed in [157] [158]. The idea is to overlap reconfiguration with computation. Since the reconfiguration can be done in background during computation, the reconfiguration overhead is reduced by that overlapping time.

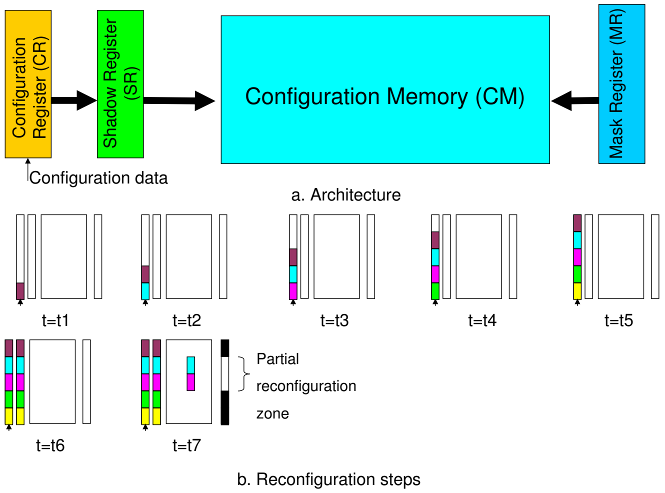


Figure 2.10: A partially reconfigurable FPGA

Changing a pipeline circuit per stage incrementally instead of a whole circuit at once is proposed in [159] [160] to reduce reconfiguration overhead. By doing so, the current computation, the next computation, and the reconfiguration can be taken place concurrently in different stages. As a result, the reconfiguration overhead is reduced.

Using multi-context FPGAs is another way to deal with long reconfiguration times as proposed by [162] and [163]. However, *n*-context FPGAs need *n* times more SRAM memory for saving configuration data. When one context is being reconfigured (passive context), the other context (active context) is used to define the FPGA fabric current operation. As a result, the FPGA keeps working during reconfiguration. The reconfiguration overhead in this case is just the time needed for switching between two contexts, which is very short and usually can be done in a single clock cycle.

Creating multi-channel configuration circuits to reconfigure FPGAs can also be used to shorten reconfiguration time [56]. Since multiple parallel configurations can be transferred, the reconfiguration overhead is reduced.

Bitstream compression has also been proposed to speedup reconfiguration when the bottleneck is in transferring data from memory to the controller to drive the configuration circuit (e.g. [164]- [170]). The basic idea is to reduce the bitstream size by doing data compression, thus reducing the time for transferring the bitstream to controller. However, the maximal configuration speed is still limited by the maximum speed of the configuration circuit [165].

Minimizing addressing overhead is another way to reduce reconfiguration time as reported in [171]. Since the address data are included in the bitstream, reducing this overhead results in reduced bitstream size. As a consequence, the reconfiguration time is reduced.

Since only configuration data that are different need to be altered in partially reconfigurable FPGAs, maximizing common configuration data between successive configurations can be used to reduce reconfiguration overhead as reported in [172], [177], [180], [185], and [186]. This technique is called as configuration reuse. The effect of circuit placement and configuration granularity on this technique is reported in [173]. Combining configuration reuse and configuration prefetching techniques to minimize reconfiguration time is presented in [175]. Reusing previously communication infrastructure to minimize reconfiguration overhead for task communications is proposed in [176].

Minimizing the number of required reconfigurations can be used to reduce reconfiguration overhead. Some techniques based on this idea are presented as follows. Loop transformations to maximize configuration reuse for reducing the number of needed reconfigurations are presented in [181]. Replacement policy to reduce the number of required reconfigurations is proposed in [161] and [184]. Replacing least recently used tasks is proposed in [161]. Besides, a credit-based replacement policy is also proposed in [161]. Every time a currently placed task is reused, its credit is set to its size. The smallest credit task is replaced if there is no room for incoming task. Replacing longest period tasks is proposed as a replacement policy in [184].

Reusing statements inside an FPGA mapped loop before reconfiguring it for the next statements proposed in [182] reduces the number of reconfigurations. Changing the execution order of hardware tasks can also be used to reduce the number of reconfigurations as reported in [183].

Minimizing the number of frames reduces bitstream size. As a result, the reconfiguration time is reduced. Some techniques based on this idea are presented as follows. Adding unused configuration frames into the cost function during routing to minimize the number of used frames is proposed in [174]. The LUT input orders are permuted such a way that the changing memory bits are located into some common frames is proposed in [178]. Placing the logic elements into as few slice columns as possible is another solution to reduce the number of frames [179]. Since the narrow implementations lead to a high switch box and routing resource utilization, this idea can reduce performance and increase power consumption as studied in [187].

Since FPGAs are fine-grained reconfigurable devices, they require a large amount of configuration bits. To reduce this configuration bits, many researchers have an idea to use coarse-grained reconfigurable devices as reviewed in [188] which require less configuration bits but suffer from lower flexibility. To reduce FPGA reconfiguration overhead, a coarse-grained reconfigurable array, called as QUKU, is implemented on an FPGA in [189]. The functionality of each element of the reconfigurable array and its interconnection can be reconfigured at very short time because it is coarse-grained. A different coarse-grained array can be build for optimizing the array for a specific application. Because of less flexible, circuits with QUKU run slower than circuits without QUKU as reported in [189].

To amortize reconfiguration overhead, in [190] [191] [192], the throughput of data transferring from memory to the configuration circuit is maximized. Since the data can be transferred faster, the circuit can work at its top speed which reduces reconfiguration time. However, again the maximum configuration speed is still bounded by the configuration circuit bandwidth [165].

Preventing larger and frequently reconfigured hardwares to be reconfigured is reported in [193] to reduce reconfiguration overhead. They used integer linear programming to determine which hardware tasks that will be assigned as fixed hardware tasks from all needed hardware tasks for a specific application targeting a given FPGA. Since these hardware tasks are fixed at runtime, the reconfiguration overhead for that application is reduced.

Merging multiple circuits into a larger and more general purpose circuit can reduce circuit area. Since circuit area determines its reconfiguration time, the reconfiguration time is reduced accordingly. This technique is reported in [194].

A two-level reconfiguration is proposed in [195]. In the first level, the configurable memory cells that need to be reconfigured are linked together in a chain; whereas the other cells that do not require reconfiguration are bypassed in the chain to speedup reconfiguration. In the second level, the configuration data are serially transferred to the chain to partially reconfigure the device.

A balanced binary tree structure is proposed in [196] for transferring configuration data to a reconfigurable device. The reconfiguration is done in two stages. The first stage is to prepare the structure for allowing fast reconfiguration with minimum address information. The second stage then uses the prepared structure to transfer the configuration data to the reconfiguration memory. In [196], the tree must be a balanced binary tree, this reduces flexibility of this proposal. To have more flexibility, the new structure for partial reconfiguration is proposed in [197] to allow unbalanced tree to be built in the device.

2.5 Summary

In this chapter, we have presented a survey on existing online hardware task scheduling and placement algorithms, techniques to reduce power consumption in reconfigurable devices, and previous work on reducing reconfiguration overhead in runtime reconfigurable systems.

Because the running time of online algorithms is considered as an overhead for the overall execution time of applications, therefore not only placement quality but also the speed of the algorithm should be addressed. Many algorithms have been proposed to deal with the scheduling and placement in runtime reconfigurable systems. However, none of them has a blocking-aware ability; the existing algorithms have a tendency to block future tasks to be scheduled earlier, referred as ”blocking-effect”. As a result, wasted area (volume), schedule time, and waiting time will increase significantly. To cope with this problem, we propose two online placement algorithms in Chapter 3 and two online scheduling and placement algorithms in Chapter 4.

Although many techniques have been proposed for power reduction in field-programmable devices (FPDs), they are all based on conventional logic elements (LEs). In the conventional LE, the output of the combinational logic (e.g. the lookup table (LUT) in many FPGAs) is connected to the input of the storage element; while the D flip-flop (DFF) is always clocked even when it is not necessary. Such unnecessary transitions waste power. To address this problem, we propose a novel low power LE as presented in Chapter 5.

All presented solutions for reducing reconfiguration overhead have a common characteristic that they do not directly target the configuration circuit architecture which is the major contributor to the high reconfiguration cost. The high reconfiguration times are due to the large amount of configuration bits sent through a constrained data path. In order to alleviate this, we propose a novel FPGA configuration circuit architecture to speedup bitstream (re)configuration and relocation as shown in Chapter 6.

3

Online Hardware Task Placement Algorithms

Online hardware task placement algorithms are expected to find the best location on a partially reconfigurable device for each arriving task in the shortest time possible. This is due to the fact that the execution time of the online placement algorithm is introducing an overhead extending the overall execution time of the applications. As a result, execution speed and placement quality are two very important attributes of a good online placement algorithm. Usually algorithms trade-off between placement quality and execution speed. In general, high placement quality algorithms are slow. On the other hand, fast algorithms have poor placement quality. Hence discovering a high quality, fast placement strategy is challenging. To address this challenge two novel solutions are proposed in this chapter.

First, we speedup existing algorithm while maintaining placement quality. Bazargan's algorithm [22] is used as a case study. Three techniques, referred as Merging Only if Needed (MON), Partial Merging (PM) and Direct Combine (DC) are proposed improve algorithm execution time. To preserve the placement quality, one strategy, Combine Before Placing (CBP), is introduced. The above techniques and strategy form the Intelligent Merging (IM) algorithm.

The second solution is to design a new, simpler algorithm with high placement quality. As mentioned above an algorithm designer usually has to trade off between placement quality and execution speed. To address this, we propose the Quad-Corner (QC) algorithm which is a simple yet effective algorithm.

This chapter is organized as follows. Our Intelligent Merging algorithm is proposed and evaluated against state of the art in Section 3.1. In Section 3.2, we introduce our novel Quad-Corner algorithm and evaluate it. Finally, Section 3.3 ends with some conclusions.

3.1 Intelligent Merging Algorithm

3.1.1 Basic Idea of the Intelligent Merging Strategy

The IM algorithm consists of three techniques (MON, PM, DC) and one strategy (CBP). To reduce the algorithm execution time, IM is equipped with the Merging Only if Needed (MON) technique that allows IM to merge blocks of empty area only if there is no available block for the incoming task. To terminate merging process earlier, IM is armed with Partial Merging (PM) technique to give it an ability to merge only a subset of the available blocks. To further reduce the algorithm execution time, IM can directly combine blocks using its Direct Combine (DC) technique. To increase the placement quality, Combine Before Placing (CBP) strategy always directly merges blocks to form a bigger block before placing a task when possible.

3.1.2 The Merging Only if Needed Technique

Merging Only if Needed (MON) is a technique where Non-overlapping Empty Rectangles (NERs) are merged only if there is no available NER for placing the arriving task. By doing so we can save algorithm execution time (the original Bazargan's algorithm always merges NERs).

Figure 3.1 shows how our MON technique works. The top left corner of Figure 3.1 depicts the empty FPGA model (the beginning status) that consists of a sin-

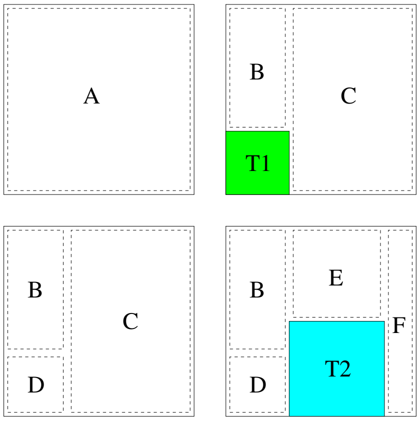


Figure 3.1: MON technique

gle NER (NER A). If there is a new task (T1), the task is placed on the bottom left of NER A. This process produces two new NERs (B and C) as shown on the top right of the same figure. The bottom left of Figure 3.1 shows the FPGA area when task T1 is removed from the FPGA after completion, leaving one new NER (NER D). In this situation, Bazargan's algorithm works differently as it would merge the NERs (NERs B, C, and D) into one single bigger NER (NER A in our example). Hence Bazargan's algorithm spends computational time on (unnecessary) merging every time a task completes. In case of MON when a new task (T2) arrives, it is placed on one of the available NERs (in our example NER C) that has enough size to accommodate it. Reducing the unnecessary merging is the key factor in our MON technique for improving the Bazargan's algorithm execution time.

3.1.3 The Partial Merging Technique

The Partial Merging (PM) technique allows our Intelligent Merging mechanism to merge only a subset of the available NERs until there is enough free space for the new task. We thus again save algorithm execution time by terminating the merging process earlier. In Bazargan's algorithm as mentioned earlier all available NERs will be merged.

Figure 3.2 shows how our PM technique works. Top left of Figure 3.2 shows

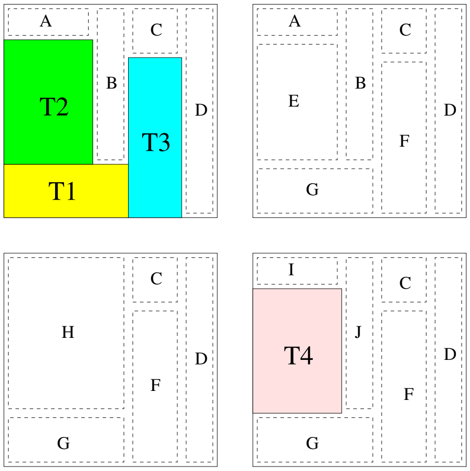


Figure 3.2: PM technique

how three tasks (tasks T1, T2, and T3) have been placed on the FPGA. Task

T2 produces two NERs (NERs A and B), while task T3 produces another two NERs (NERs C and D). The top right of Figure 3.2 shows the situation when these three tasks are removed from the FPGA and three new NERs (NERs E, F, and G) become available. Let us assume task T4 arrives and has to be placed. At this point, there is no single NER available that can fit this new task. In this case, IM needs to merge NERs and form a bigger NER for this new task. In order to accommodate task T4 (bottom right of Figure 3.2), the PM technique in our IM algorithm only needs to perform one merge operation (NERs A, B, and E) and form a new bigger NER (NER H) (bottom left of Figure 3.2). Again, Bazargan's algorithm would perform additional merging. More precisely, the merging of NERs A, B, and E, then merging C, F, and D, and finally merging of all of them into one new bigger NER is required. In this example, Bazargan's algorithm needs three merging operations while our IM needs only one. We call this technique also *merge-on-demand* which is the key element of the proposed PM technique to reduce algorithm execution time.

3.1.4 Direct Combine and Combine Before Placing

The Direct Combine (DC) technique allows IM to combine NERs directly without merging and splitting operations, thereby saving algorithm execution time. Figure 3.3 shows how the proposed DC technique works.

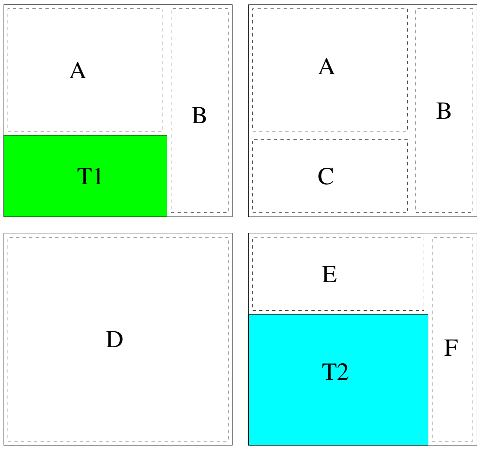


Figure 3.3: DC technique

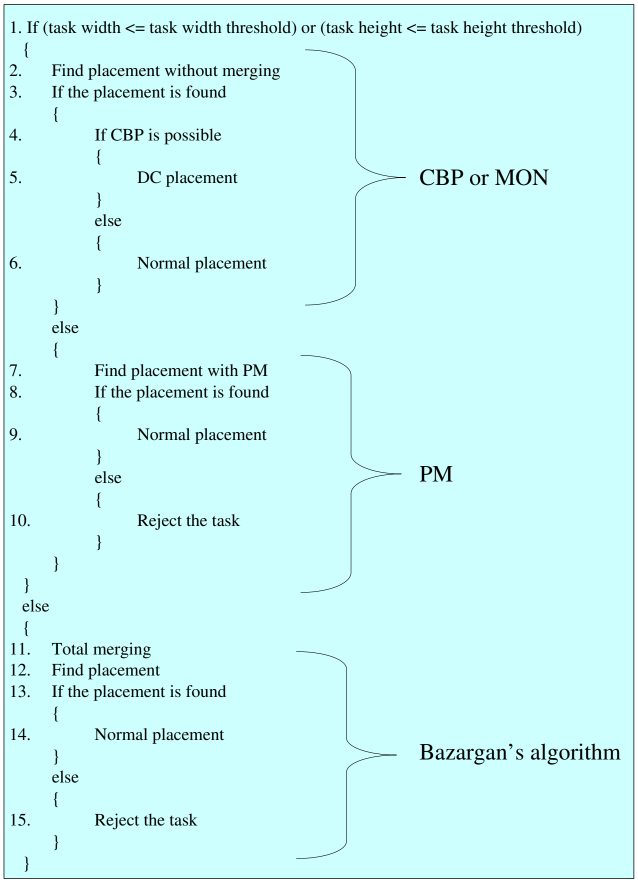
The top left of Figure 3.3 shows the beginning situation when a task T1 is placed on the FPGA. This leads to two NERs (NERs A and B). The top right of Figure 3.3 shows the FPGA after T1 has been completed. The new NER (NER C) is produced. Let us assume that task T2 arrives. At this point, all NERs in this location are free, so it is possible to merge the NERs (NERs A, B, and C) to form a new bigger NER (NER D) as in the Bazargan algorithm. To decrease algorithm execution time, instead of merging (release memory) and splitting (allocate memory) NERs, the DC technique directly combines the NERs (NERs A, B, and C) to create a bigger NER (NER D) (bottom left of Figure 3.3). The resulting NER can be used to place the new task (bottom right of Figure 3.3). To increase the placement quality, the DC technique will always directly combine NERs into a bigger NER before placing new tasks. We call this Combine Before Placing (CBP) strategy. For example if the size of task T2 on Figure 3.3 is smaller than NER A, the DC technique will not directly place the task on NER A. To prevent fragmentation, our DC technique will combine these three empty NERs (NERs A, B, and C) before placing the task on this newly formed NER. Therefore the CBP strategy decreases the fragmentation of empty areas and increases the placement quality.

3.1.5 Intelligent Merging Algorithm

To speedup the execution time of Bazargan's algorithm without loosing its good placement quality, we propose to dynamically combine the above three techniques (MON, PM, DC) and our CBP strategy for small to medium task sizes. If the task is too large, the possibility that the task can be placed without merging decreases, so in this case our techniques will not work. Therefore, IM will conditionally activate our techniques and strategy based on the task sizes as shown in Figure 3.4.

If the task is not too large (the task width $\leq$ the task width threshold or the task height $\leq$ the task height threshold), IM will do CBP or MON (lines 2-6). If IM fails to find placement after doing CBP or MON, IM will do PM (lines 7-10). If IM also fails to find placement after doing PM, IM will reject the task. If IM can find placement using CBP (line 4), IM will place the task using DC placement (line 5). If IM can find placement using MON, IM will place the task using normal placement (line 6).

If the task is too large, IM will do full merging before finding placement like the original Bazargan's algorithm (lines 11-15). If IM can find placement after complete merging, IM will place the task using normal placement (line 14), otherwise IM will reject the task (line 15).



```
If (task width <= task width threshold) or (task height <= task height threshold)
  {
    Find placement without merging
    If the placement is found
    {
        If CBP is possible
        {
            DC placement
        }
        else
        {
            Normal placement
        }
    }
    else
    {
        Find placement with PM
        If the placement is found
        {
            Normal placement
        }
        else
        {
            Reject the task
        }
    }
  }
  else
  {
    Total merging
    Find placement
    If the placement is found
    {
        Normal placement
    }
    else
    {
        Reject the task
    }
  }
```

Figure 3.4: Pseudocode of Intelligent Merging algorithm

3.1.6 Evaluation

Experimental Setup

We have constructed a discrete-time simulation framework in ANSI-C to evaluate the performance of the proposed techniques and algorithm and compare it to related work. Our experiments have been conducted on a Pentium-IV 3.4 GHz work station. Each task is placed at its arrival time and in cases the placement fails, it is assumed rejected. In other words, there is no-queue used for the scheduling. Only one new task can arrive at each simulated time unit. Furthermore, our scheduling scheme is non-preemptive – once a task is loaded onto the device it runs to completion.

We model an FPGA with size of 100x100 reconfigurable units and use tasks with randomly generated sizes and life-times. To our best knowledge, there are no standard benchmarks available to evaluate online placement algorithms. Therefore, we generated our own synthetic benchmark sets. To closely approximate real-life scenarios, we generate randomly 13 task sets as depicted in Table 3.1 ranging from short life-time tasks (50 time units) till long life-time tasks (200 time units) and also from small size tasks (4 reconfigurable units) till large size tasks (400 reconfigurable units). The last task set is a mixed task set (MTS) of TS1 to TS12. *Wmin*, *Wmax*, *Hmin*, *Hmax*, *Ltmin*, and *Ltmax* denote minimum task width, maximum task width, minimum task height, maximum task height, minimum life-time and maximum life-time respectively. Our task sets consists of 1000 tasks with uniformly distributed life-times and task sizes.

Using our simulation framework, we compared our IM algorithm to Bazargan's proposal [22]. In this simulation, we set the task width threshold=10 and the task height threshold=10 for our algorithm. For Bazargan's algorithm, we use the First Fit (FF) heuristic for choosing NERs and the Shorter Segment (SSEG) heuristic for splitting decision, because these heuristics provide the best performance, as mentioned in [22].

Table 3.1: Simulated task sets (W:task width, H:task height, Lt:task life-time)

Task Set	Wmin	Wmax	Hmin	Hmax	Ltmin	Ltmax
TS1	2	5	2	5	50	100
TS2	2	5	2	5	100	150
TS3	2	5	2	5	150	200
TS4	5	10	5	10	50	100
TS5	5	10	5	10	100	150
TS6	5	10	5	10	150	200
TS7	10	15	10	15	50	100
TS8	10	15	10	15	100	150
TS9	10	15	10	15	150	200
TS10	15	20	15	20	50	100
TS11	15	20	15	20	100	150
TS12	15	20	15	20	150	200
MTS	2	20	2	20	50	200

Our study is based on evaluation of two performance parameters: the average percentage of accepted tasks (%) and the average algorithm execution time(μs). A task is considered accepted if the algorithm can successfully find a place for running that task on the reconfigurable device. The percentage of accepted tasks is the ratio between the number of accepted tasks and the total number of tasks. The execution time was obtained using *gettimeofday()* function that provides us with microseconds precision. We define the algorithm execution time as the time used by the algorithm for a single task placement. Good quality placement algorithms have higher percentage of accepted tasks in general. The average percentage of accepted tasks represents the average placement quality, while the average algorithm execution time is a metric for the algorithm performance. The average values are obtained after 1000 algorithm iterations for each task set.

To study the impact of the different techniques, we performed experiments with five different cases:

- BFFSSEG: Bazargan's algorithm using FF and SSEG heuristics [22];
- MON: algorithm using MON technique;
- MON+PM: algorithm using combination of MON and PM techniques;
- MON+PM+DC: combination of MON, PM, and DC techniques;
- IM: our Intelligent Merging algorithm.

Experimental Results

The average accepted tasks precentage for each task set is depicted in Figure 3.5. The effect of each technique on the number of accepted tasks is shown in Figure 3.6. Positive values mean increase of the number of accepted tasks, while negative values mean decrease in accepted tasks number. This figure is obtained by comparing the results of the algorithm with and without applying each proposed technique. The average algorithm execution time over 1000 runs for each task set is depicted in Figure 3.7. The effect of each technique on the algorithm execution time is shown in Figure 3.8.

Effect of task size and life-time

As the task size increases, the average percentage of accepted tasks decreases because it is more difficult to find available free space that can accommodate the task. Longer task life-times decrease the average percentage of accepted tasks, because the task will stay longer on the FPGA. It is thus more difficult to find available free space that can accommodate other tasks.

Large tasks negatively influence the algorithm execution time. This is to be expected, because when the task size is bigger, the possibility that the task can be placed on one of the NERs or combined NERs on the FPGA without merging decreases. A similar observation holds for the life-time of tasks where

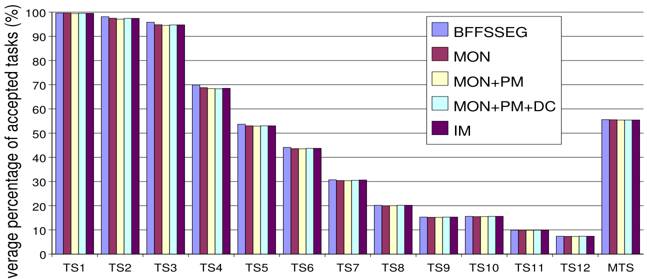


Figure 3.5: Average percentage of accepted tasks (%)

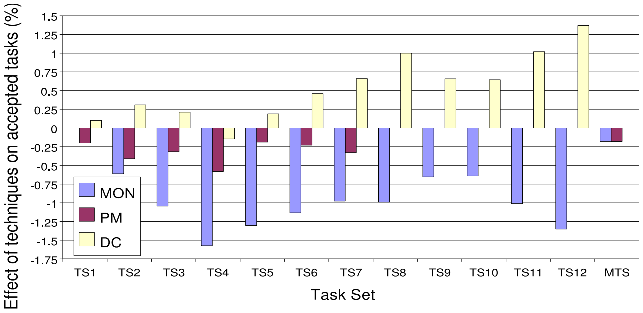


Figure 3.6: Effect of techniques on accepted tasks(%)

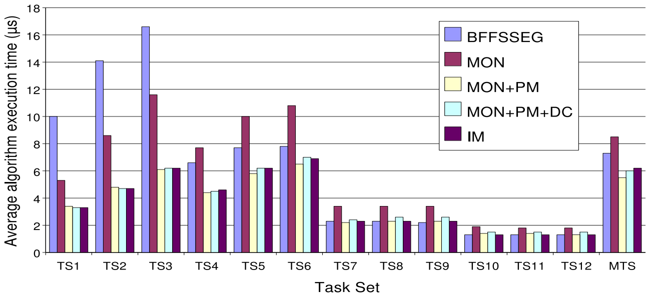


Figure 3.7: Average algorithm execution time(μs)

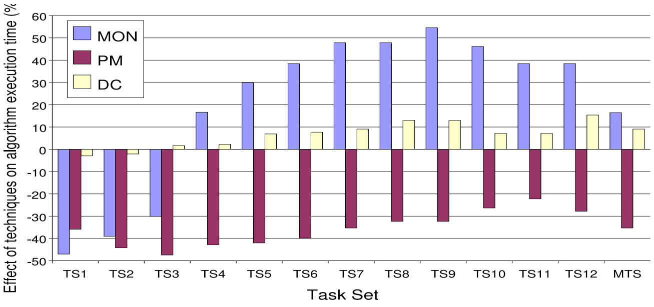


Figure 3.8: Effect of techniques on the algorithm execution time(%)

the execution time is negatively influenced as tasks will stay longer on the FPGA. Therefore the probability that subsequent tasks can be placed on one of the NERs without merging becomes smaller.

Evaluation of algorithm using MON technique

The algorithm using the MON technique is up to 1.9 times faster than the Bazargan's algorithm due to intelligently avoiding full merging with similar accepted task percentage. On average, the number of accepted tasks is reduced by 0.95 %. However for the mixed task set, this is only 0.18 %.

Evaluation of algorithm using combination of MON and PM techniques

Among these algorithms, the algorithm using a combination of MON and PM techniques (MON+PM) performs the best in terms of algorithm execution time on average. The algorithm is up to 2.9 times faster than the Bazargan's algorithm with similar accepted tasks as the result of intelligently avoiding total merging and exploiting its merge-on-demand capability. On the average, the decrease of accepted tasks is 1.24 %. However for the mixed task set, the decrease is only 0.36 %.

Evaluation of algorithm using combination of MON, PM, and DC techniques:

The algorithm using combination of MON, PM, and DC techniques (MON+PM+DC) is up to 3 times faster than the Bazargan's algorithm with similar accepted tasks as the result of intelligently avoiding total merging and exploiting its merge-on-demand and direct combine capability. On the average, the decreasing of accepted tasks is 0.95 %. However for mixed task set, the decreasing is only 0.36 %.

Evaluation of IM algorithm

IM can effectively exploit the advantages of our three techniques especially when the tasks are not too large, because the possibility that the tasks can be placed without merging become large.

The IM algorithm is up to 3 times faster than the Bazargan's algorithm with similar accepted tasks by intelligently exploiting the proposed three techniques. On the average, the decreasing of accepted tasks is 0.89 %. However for the mixed task set, the decreasing is only 0.36 %.

On the basis of these results, we can state that our algorithm produces comparable results as Bazargan with a slight minor difference for the worst case but similar placement quality in the best case.

Effect of the MON technique

The MON technique can decrease the algorithm execution time for small task sets. When the tasks are small, the possibility that the tasks can be placed on one of NERs without merging becomes bigger, so in this case MON can prevent total merging effectively.

The MON technique decreases up to 47 % of the algorithm execution time by intelligently avoiding total merging. On the average, the MON technique decreases 0.95 % accepted tasks. However for the mixed task set, the decrease in only 0.18 %.

Effect of the PM technique

The PM technique decreases the algorithm execution time up to 47.4 % due to the merge-on-demand. On the average, the PM technique decreases 0.29 % accepted tasks. However for the mixed task set, the decrease is only 0.18 %.

Effect of the DC technique

The DC technique decreases the algorithm execution time for sets with small task sizes, as the possibility for task placement on one of the combined NERs without merging increases. This improves the DC technique efficiency.

We see that the DC technique increases the number of accepted tasks for almost all task sets except TS4 as the result of its CBP strategy.

The DC technique decreases up to 2.94 % algorithm execution time by intelligently avoiding merging and splitting. On the average, the DC technique decreases 0.29 % accepted tasks. However for the mixed task set, it does not affect on accepted tasks.

3.2 Proposed Quad-Corner Algorithm

3.2.1 Basic Idea of Quad-Corner Strategy

The existing strategies tend to place arriving tasks concentrating on one corner and (or) split free area into many small segments as shown in Figure 3.9a. These can lead to the undesirable situation that a task cannot be allocated even if there would be sufficient free area available. Because of these problems, task T5 cannot be accommodated as shown in this motivating example in Figure 3.9a. As a consequence, the reconfigurable device is not well utilized (waste of resources). Furthermore, task T5 has to wait for execution or to be executed by the host processor due to this inefficient placement, hence the application will be slowed down (performance degradation). To address these problems, we spread hardware tasks close to the four corners of the devices as illustrated in Figure 3.9b. There are two main advantages of this strategy as shown in Figure 3.9b: (1) it reserves a lot of free area in the middle of the device; (2) it solves splitting free area problem. As a result, both the reconfigurable device utilization and the system performance will be increased.

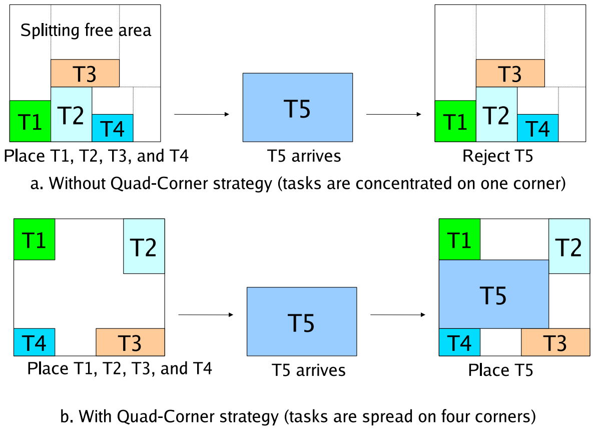


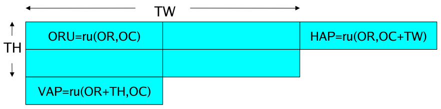
Figure 3.9: Basic idea of quad-corner strategy

3.2.2 Two-dimensional Reconfigurable Device

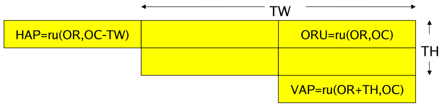
A two-dimensional reconfigurable device, denoted as *RD(H,W)*, consists of *HxW* homogeneous reconfigurable units arranged in a two-dimensional array of height *H* and width *W* and an interconnect between the units. A reconfigurable unit in row *r* and column *c* is represented by *ru(r,c)*, for $0 \leq r \leq H-1$ and $0 \leq c \leq W-1$, with *ru(0,0)* as the upper left corner.

3.2.3 Task Types

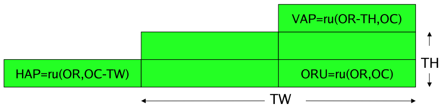
To make the free area in the middle of the device as large as possible, we force our algorithm to spread hardware tasks close to the four corners of the devices. To support this idea, we define four ways of placing tasks: starting from upper left corner, upper right corner, lower right corner, and lower left corner. We divide tasks into four different task types (Figure 3.10): upper left task (ULT), upper right task (URT), lower right task (LRT), and lower left task (LLT). A *THxTW* task in a two-dimensional reconfigurable device *RD(H,W)* is a group of reconfigurable units belonging to *RD(H,W)*, with task height *TH* and task width *TW*, such that $1 \leq TH \leq H$ and $1 \leq TW \leq W$. The task has an origin reconfigurable unit *ORU=ru(OR,OC)* and two alternative placement positions for accommodating future tasks. The two alternative positions are a *horizontal alternative position* (HAP) and a *vertical alternative position* (VAP) as origin reconfigurable units for future tasks. OR and OC denote the origin row and origin column respectively.



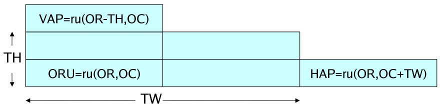
a. A 2x2 upper left task (ULT)



b. A 2x2 upper right task (URT)



c. A 2x2 lower right task (LRT)



d. A 2x2 lower left task (LLT)

Figure 3.10: Examples of four task types and their alternative placement positions

3.2.4 Initial Placement Positions

To spread tasks to the corners for creating as large as possible free area in the middle, we propose four initial placement positions for accommodating tasks on an empty *WxH* two-dimensional reconfigurable device: upper left task initial *ULTI=ru(0,0)* (for upper left tasks), upper right task initial *URTI=ru(0,W-1)* (for upper right tasks), lower right task initial *LRTI=ru(H-1,W-1)* (for lower right tasks), and lower left task initial *LLTI=ru(H-1,0)* (for lower left tasks).

3.2.5 Data Structures

We use a 2D matrix *RD(r,c)* (RD matrix) to represent the FPGA area, defined as: *RD(r,c)=0* if *RD(r,c)* is not occupied (free) or *RD(r,c)=1* if *RD(r,c)* is occupied (used), where $0 \leq r \leq H-1$, $0 \leq c \leq W-1$, and *RD(0,0)* is the element of upper left corner.

During placement, the software implementation of the proposed algorithm maintains four lists: an upper left task list (for storing upper left tasks), an upper right task list (for storing upper right tasks), a lower right task list (for storing lower right tasks), and a lower left task list (for storing lower left tasks).

3.2.6 Searching Sequences for Placement

To accommodate tasks, the algorithm needs to search four task lists as mentioned above. In order to pack tasks more compactly, the algorithm searches placements in all different task lists based on the sizes of arriving tasks. There are four different searching sequences for placement: upper left corner first (for very large tasks), upper right corner first (for large tasks), lower right corner first (for medium size tasks), and lower left corner first (for small tasks). The strategy tries to group tasks based on their sizes. This way, the algorithm picks the corner which contains tasks that are similar in size as the task that needs to be placed. For example, finding placements for very large tasks using upper left corner first sequence are fastest. The reason for this is that the algorithm finds a placement starting from the location where very large tasks were mapped. This strategy can also increase the placement quality since we group tasks based on their sizes for better compacting purposes. In this thesis, we consider serial implementation of the list search. Since the four task lists can work independently, searching task lists can be executed in parallel in a future implementation.

3.2.7 The Algorithm

The pseudocode of the proposed Quad-Corner algorithm for allocation is shown in Figure 3.11a. In line 1, the algorithm searches dynamically different possible placements according to its size until it finds the appropriate placement. This strategy reduces the algorithm execution time and at the same time increases its placement quality by finding placements in the specific area and placing tasks as close as to the specific group. If the algorithm finds the placement position, it places the task starting from this position by updating the RD matrix (line 3) and adds the task to its corresponding task list (line 4). If the algorithm cannot find the placement, it rejects the task (line 5).

```
1. Do searching sequences
   for placement
2. If the placement is found
   {
3.    Place the task by
      updating RD matrix
4.    Add the task to the
      corresponding task list
   }
5. Else reject the task
```

a. Allocation

```
1. If the life-time of the
   task is zero
   {
2.    Delete the task
      from RD matrix
3.    Delete the task from
      the corresponding
      task list
   }
```

b. Deallocation

Figure 3.11: The pseudocode of QC strategy

The pseudocode of the proposed Quad-Corner algorithm for deallocation is shown in Figure 3.11b. In line 1, the algorithm checks the life-time of the task. If the life-time is zero (finished tasks), the algorithm deletes the task from RD matrix (line 2) and its corresponding task list (line 3).

3.2.8 Evaluation

Experimental Setup

To evaluate the proposed algorithm, a discrete-time simulation framework was constructed in C. The framework was compiled and run under Linux on a Pentium-IV 3.4 GHz PC. Since the algorithms are online, the information about new tasks is unknown until their arrival time. We assume that each task should be placed at its arrival time and is rejected when it could not be placed. If a task is rejected, the equivalent function should be executed in software by the host processor and hence a penalty is incurred. We use task set $REJECT$ to represent tasks which are rejected from all task set TS. The volume of a task t_i that has a width w_i reconfigurable units, height h_i reconfigurable units and life-time lt_i time units is defined as $v_i(t_i) = w_i.h_i.lt_i$. For simplicity but without loss of generality, we assume the penalty to be linearly proportional to the volume of the rejected task. The *penalty ratio* is the ratio between the total volume of rejected tasks ($\sum_{\forall t_i \in REJECT} v_i(t_i)$) and the total volume of all tasks ($\sum_{\forall t_i \in TS} v_i(t_i)$). When a task is rejected, the total free area in the reconfigurable device is called the wasted area. The *wasted area ratio* is the ratio between the wasted area and the total area of the reconfigurable device. Generally, algorithms with a higher placement quality will exhibit lower penalty and wasted area ratios.

We evaluated the QC algorithm using real hardware tasks on a real FPGA. We use the benchmark set from [54] (e.g. MDCT, matrix multiplication, hamming code, sorting, FIR, ADPCM, etc) and use the DWARV [55] C-to-VHDL compiler to translate the benchmarks to VHDL. The benchmarks are synthesized with the Xilinx ISE 8.2.01i_PR_5 tools [198] [199] targetting Virtex-4 XC4VLX200 device with 116 columns and 192 rows of reconfigurable units. From these hardware implementations, we obtain the required resources, the reconfiguration times and the execution times of the hardware tasks. Some examples of implemented hardware tasks are shown in Table 3.2. For example, the area A_i for function POWER obtained after synthesis is 444 CLBs. In [56], one Virtex-4 row has a height of 16 CLBs. By choosing two rows for implementing this function, we obtain $h_i = 2\text{x}16 = 32$ CLBs and $w_i = \lceil A_i/h_i \rceil = \lceil 444/32 \rceil = 14$ CLBs. The function needs 37 cycles with 11.671 ns clock period (85.68 MHz). Hence, we estimate the execution time of 100 back-to-back operations to be $et_i = 37\text{x}11.671\text{x}100 = 43183$ ns. There are 22 frames per column and each frame contains 1312 bits. Therefore one column needs $22\text{x}1312 = 28864$ bits. Since the function occupies 14 CLBs in 2 rows (32 CLBs), we obtain a bitstream with $14\text{x}2\text{x}28864 = 808192$ bits. Since ICAP can send 32 bits per clock cycle at 100 MHz, we estimate the reconfiguration time $rt_i = 808192\text{x}10/32 = 252560$ ns. In the simulation, we assume that the life-time lt_i is the sum of reconfiguration time rt_i and execution time et_i. The hardware tasks are selected randomly from 37 implemented hardware tasks. Every task set consists of 100 tasks, each of which has a life-time and task size. Since we target runtime dynamic multitasking multiuser systems which hardware tasks can arrive any time, the arrival periods of hardware tasks are randomly generated between 10 μs to 20 μs, 20 μs to 30 μs, and 30 μs to 40 μs.

Table 3.2: Some examples of implemented hardware tasks (et_i for 100 operations, rt_i at 100 MHz)

No.	Hardware Tasks	w_i (CLBs)	h_i (CLBs)	et_i (ns)	rt_i (ns)
1	functionPOWER	14	32	43183	252560
2	adpcm_decode	10	32	770302	180400
3	adpcm_encode	10	32	1031213	180400
4	FIR	33	32	1565980	595320
5	mdct_bitreverse	32	64	449412	1136520
6	mmul	25	64	57278	892980

Because our algorithm is a first fit (FF) heuristic algorithm that can find placements for arriving tasks very fast, we compare the algorithm only with FF heuristic algorithms which are faster than best fit (BF) heuristic algorithms. To fairly evaluate the algorithms, we use the version of Bazargan's algorithm with the best placement quality, i.e. using the FF heuristic for choosing non-overlapping empty rectangles (NERs) and the Shorter Segment (SSEG) heuristic for splitting, as mentioned in [22]. In addition, we also compare the proposed algorithm to Intelligent Merging (IM) algorithm (the faster modified version of Bazargan's algorithm).

Experimental Results

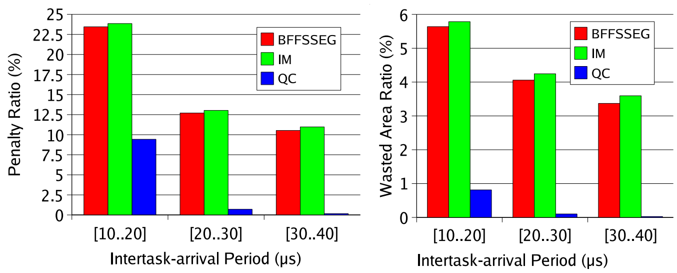
Results were obtained using the aforementioned discrete-time simulation framework and by comparing the following algorithms: Bazargan's (BFF-SSEG) algorithm [22], Intelligent Merging (IM) algorithm, and Quad-Corner (QC) algorithm as presented in Figure 3.12. Average numbers are obtained by running the algorithms 10000 times for every task set.

A longer inter-task arrival period creates more possibilities for additional running tasks to be finished before the arrival of new tasks. As a consequence, the penalty and wasted area are reduced as the inter-task arrival period increases.

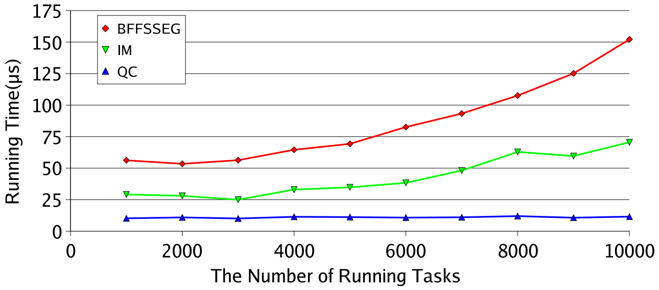
Due to its on-demand merging, the IM algorithm runs faster than the Bazargan's algorithm with almost similar penalty ratio and wasted area ratio. The Bazargan's and IM algorithms do not perform well because of splitting and fragmentation problems. Figure 3.12a shows that the QC reduces 78 % penalty and 93 % wasted area of the other related approaches on average by solving above problems.

The increase of the total number of running tasks creates more fragmentation of the free area. As presented earlier, the algorithms that use splitting and merging (Bazargan's and IM algorithms) in managing free area need to merge free area for accommodating arriving tasks. Therefore these algorithms need more merging operations by the increase of the number of running tasks. As a consequence, the algorithms (excluding the QC) run slower when the number of running tasks increases. Therefore, QC is more scalable in terms of runtime overhead than the other algorithms. By totally avoiding merging and its simplicity, QC not only has better placement quality but also runs faster than the other algorithms.

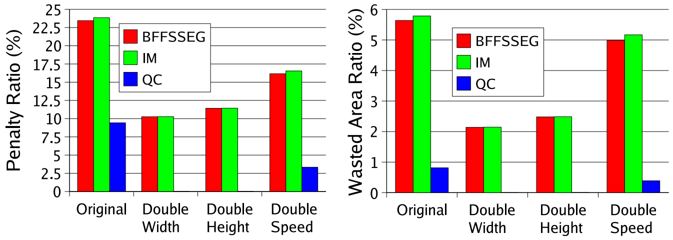
Besides comparing the performance of the algorithms using an original Virtex-4 device (Original), we also measure the effect of doubling the width (Double



a. Evaluation of placement quality



b. Evaluation of running time



c. Effect of FPGA sizes and reconfiguration speeds

Figure 3.12: Evaluation with real hardware tasks

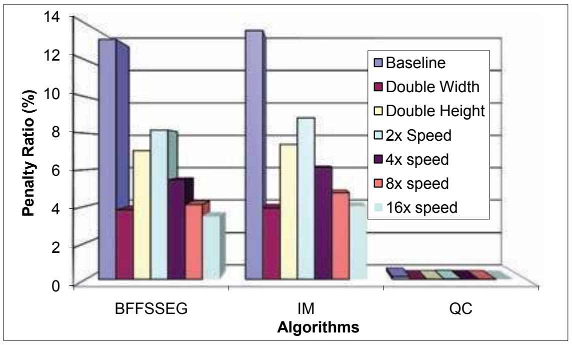
Width), the height of FPGA (Double Height), and the reconfiguration speed (Double Speed) as depicted in Figure 3.12c.

Expanding the size of the FPGA reduces the penalty and wasted area. The reason is obvious that the larger the FPGA, the easier it is to accommodate hardware tasks. As a consequence, the penalty and wasted area are decreased.

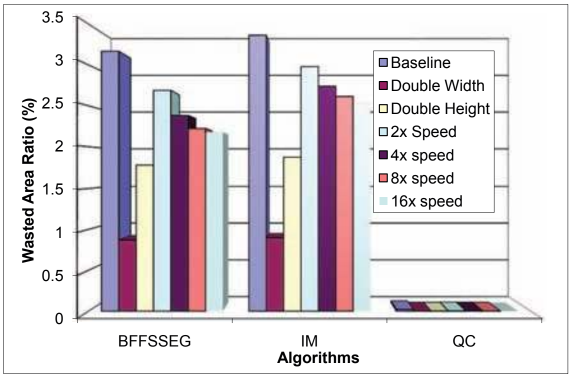
Speeding up the reconfiguration also reduces the wasted area and penalty ratios. The reason is that the faster the reconfiguration affects on less life-time of the tasks. As a result, the penalty and wasted area are dropped since the tasks stay shorter on the FPGA.

The effect of doubling the width of FPGA is more beneficial than doubling the height. The reason is that the width of the original FPGA (Virtex-4 XC4VLX200) is smaller than its height. As a consequence, doubling the width of the FPGA becomes more efficient.

Doubling the FPGA size improves placement performance more than doubling the reconfiguration speed. The effect of reconfiguration speed improvement also depends on the ratio between reconfiguration time and task execution time. The above is in agreement with the trend observed in industry that puts more pressure on increasing the FPGA area than on making the reconfiguration circuit faster.



a. Penalty ratio (%)



b. Wasted area ratio (%)

Figure 3.13: FPGA technology impact on penalty (%) and wasted area (%)

The effect of FPGA sizes (double width, double height) and reconfiguration speeds (2x , 4x, 8x, and 16x faster) on penalty and wasted area ratios (%) compared to the baseline FPGA is shown on Figure 3.13. All algorithms can benefit from bigger FPGAs or improved reconfiguration speeds (in both, time and area). Our QC algorithm is significantly better and hence can not benefit much from either FPGA size or configuration speed improvements for the test task sets. In a real system this will certainly change. These figures indicate that placement quality can be improved by using a more efficient algorithm, a faster reconfiguration circuit or a bigger size FPGA.

From the above we determine three ways to improve runtime reconfiguration systems performance. The first solution is to utilize more efficient algorithms to manage the reconfigurable resources. Therefore, it is important to study how to manage these hardware resources. The second solution is to increase the size of targeted FPGA. This solution is simple but will increase power consumption. Therefore, when this path is chosen power consumption has to be addressed. This finding triggers us to study techniques to reduce power consumption in reconfiguration devices. The third solution is to speedup the reconfiguration process requiring further study on its overhead reduction.

3.3 Summary

We proposed and evaluated two algorithms (Intelligent Merging and Quad-Corner) for online placement of reconfigurable hardware tasks. The main difference between the Intelligent Merging (IM) algorithm and state of the art is its ability to do on-demand merging. IM speeds up online placement algorithms by 1.72x while loosing only 0.89 % in placement quality. Our Quad-Corner (QC) algorithm differs from and related work by the quad-corner task distribution and its dynamic searching sequences. Spreading hardware tasks to the four corners of the devices, finding placements in the specific places, and grouping tasks in free area based on their sizes are the main key features of our proposed QC algorithm. Experiments with real hardware tasks on Virtex-4 show that the QC not only has 78 % less penalty and 93 % less wasted area than the existing algorithms on average, but also has lower runtime overhead.

Note. The content of this chapter is based on the the following papers:

T. Marconi, Y. Lu, K.L.M. Bertels, G.N. Gaydadjiev, **Intelligent Merging Online Task Placement Algorithm for Partial Reconfigurable Systems**, Proceedings of Design, Automation and Test in Europe (DATE), March 2008.

T. Marconi, Y. Lu, K.L.M. Bertels, G.N. Gaydadjiev, **A Novel Fast Online Placement Algorithm on 2D Partially Reconfigurable Devices**, Proceedings of the International Conference on Field-Programmable Technology (FPT), December 2009

4

Online Hardware Task Scheduling and Placement Algorithms

The online hardware task scheduling and placement algorithms have to find a block of hardware resources for running each arriving task on a 2D partially reconfigurable device. When there are no available resources for allocating the hardware task at its arrival time, the algorithms have to schedule the task for future execution. Here, the algorithms need to find the earliest starting time and free space for executing the task on the device in the future. Since the algorithms need to take decisions at runtime; therefore, the algorithm execution time is computed as an additional time for the overall application time. As a result, the goal of the algorithms are not only to get better scheduling and placement quality but also to have a low runtime overhead.

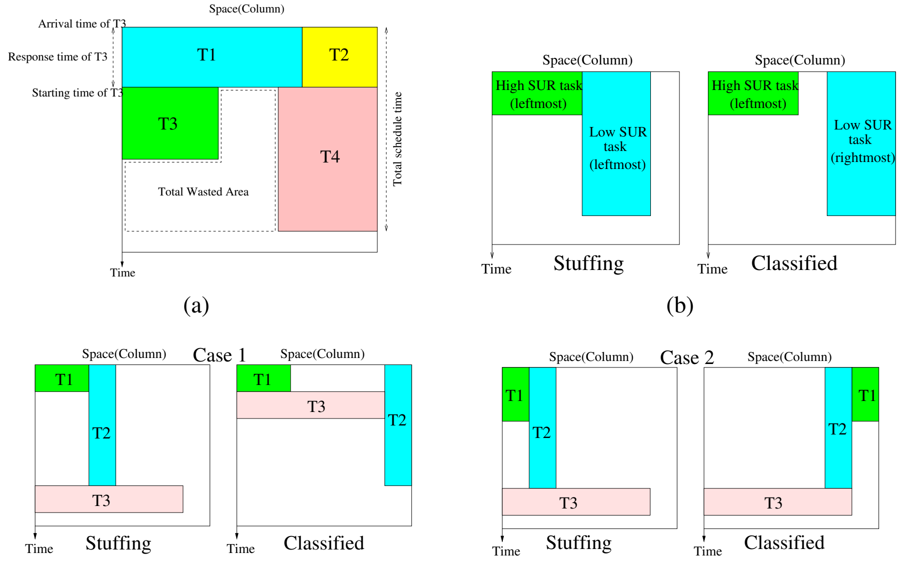
In this chapter, we propose two novel algorithms for dealing with online task scheduling and placement. The first algorithm, Intelligent Stuffing, is proposed for solving a number of shortcomings of existing algorithms for 1D area model. The second algorithm, 3D Compaction (3DC), is designed for solving ”blocking-effect” in existing algorithms for 2D area model; the algorithms tend to allocate tasks at positions where can block future tasks to be scheduled earlier. A novel 3D total contiguous surface (3DTCS) heuristic is proposed for equipping our scheduling and placement algorithm with blocking-awareness.

This chapter is organized as follows. Our Intelligent Stuffing algorithm for online task scheduling and placement targeting 1D area model of partially reconfigurable devices is proposed and evaluated against related work in Section 4.1. In Section 4.2, we introduce our 3D Compaction algorithm targeting 2D area model and then give its evaluation against related art. Finally, Section 4.3 ends with the conclusions.

4.1 Intelligent Stuffing Algorithm for 1D Area Model

4.1.1 1D Area Scheduling and Placement Problems

Given a task set representing a multitasking application with their arrival times a_i, execution times e_i and widths w_i, online task scheduling and placement algorithms targeting the 1D area model of partially reconfigurable devices have to determine placements and starting times for the task set such as there are no overlaps both in space and time among all tasks. The goals of the algorithms are: a) to utilize effectively the available FPGA resources (referred as minimize wasted area); b) to run the overall application on FPGA faster (minimize schedule time); c) to shorten waiting time of the tasks to be executed on the FPGA (minimize response time) and d) to keep the runtime overhead low (minimize the algorithm execution time).



(a) (b) (c) (d)

Figure 4.1: Performance parameters and previous algorithms

We define total wasted area as the overall number of space-time units that are not utilized as shown in Figure 4.1(a). Total schedule time is the total number of time units for the execution of all tasks. Response time is the difference between starting and arrival times for each task (in time units). Total response time is the sum of response times for all tasks. The overall algorithm execution time is the cumulative time needed to schedule and place all the tasks.

4.1.2 Intelligent Stuffing Algorithm Main Properties

In [46] [47], Steiger et al. proposed the Stuffing. It schedules tasks to arbitrary free areas that will exist in the future, including areas that will be used later by tasks currently in its reservation list. It always places a task on the leftmost of its free space as shown on Figure 4.1(b). Because the Stuffing algorithm always places tasks on the leftmost edge of the available area, it places tasks T1 and T2 as shown on Figure 4.1(c). These placements block task T3 to be scheduled earlier. In this case, it fails to place task T3 earlier.

In [48], Chen and Hsiung proposed the Classified Stuffing to solve the drawback of the Stuffing in case 1 (Figure 4.1(c)). The main difference between the algorithm and the Stuffing is the classification of tasks. It can place a task on the leftmost or rightmost of its free space based on the task Space Utilization Rate (SUR). SUR is the ratio between the number of columns required by the task and its execution time. High SUR tasks (SUR > 1) are placed starting from the leftmost available columns of the FPGA space, while low SUR tasks (SUR ≤ 1) are placed from the rightmost available columns as shown in the right of Figure 4.1(b). In case 1, it can recognize the difference between tasks T1 (high SUR task) and T2 (low SUR task), so it places successfully tasks on different placements. This makes task T3 earlier scheduling possible. However in case 2 (Figure 4.1(d)), it fails to solve the problem of the Stuffing. Because it does not recognize the difference between tasks T1 and T2 (both of the tasks are low SUR tasks), it fails to place tasks on different placements. These placements block task T3 to be scheduled earlier. Therefore in case 2, both of the previous algorithms fail to schedule task T3 earlier. Total wasted area, total schedule time, and total response time will increase as a consequence.

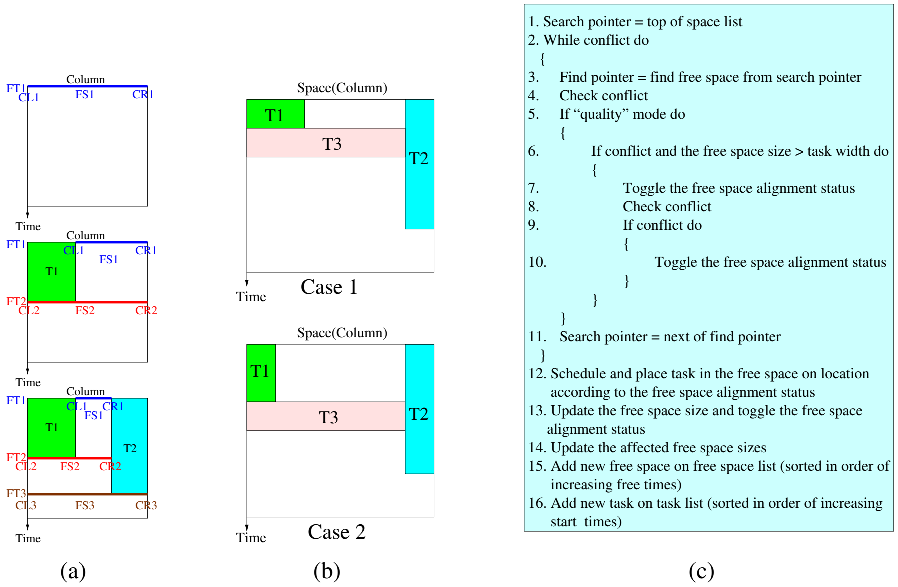
The main difference between our Intelligent Stuffing algorithm and previously proposed algorithms is the additional alignment status of the free space segments and its handling. This status guides our algorithm to make the correct decision on task placement position in order to maximize the free space area and allow earlier placing of further tasks. In addition, our algorithm does not need to compute SUR, therefore it runs faster than the Classified Stuffing.

4.1.3 The Proposed Algorithm

Our algorithm maintains two linked lists: a free space list (SL) and a task list (TL). The SL contains all free spaces FS_i with their previous pointers PP_i, dimensions (CL_i and CR_i), free times FT_i, alignment statuses AS_i and next pointers NP_i. The free time is the time when the corresponding free space can be used. The alignment status is a boolean determining the placement location of the task (leftmost or rightmost) within this free space segment. The new list entries of SL are inserted in order of increasing free times.

The TL stores all scheduled tasks with their previous pointers PP_j, start times ST_j, task dimensions (CL_j, CR_j), task execution times ET_j and next pointers NP_j. The start time is the time that the task initiates execution on the FPGA. The column left (CL_j) and right (CR_j) determine the FPGA area that is used by the task. The new list entries of TL are inserted in order of increasing of start times.

Figure 4.2(a) (top) shows an empty FPGA and a leftmost alignment status is defined, e.g., a new free space will be allocated at the leftmost position. At this point, the free space list SL contains only a single free space (FS_1) defined by its leftmost column (CL_1), its rightmost column (CR_1) and free time FT_1.



```
1. Search pointer = top of space list
2. While conflict do
   {
3.     Find pointer = find free space from search pointer
4.     Check conflict
5.     If "quality" mode do
       {
6.         If conflict and the free space size > task width do
           {
7.             Toggle the free space alignment status
8.             Check conflict
9.             If conflict do
               {
10.                Toggle the free space alignment status
               }
           }
       }
11.    Search pointer = next of find pointer
   }
12. Schedule and place task in the free space on location
    according to the free space alignment status
13. Update the free space size and toggle the free space
    alignment status
14. Update the affected free space sizes
15. Add new free space on free space list (sorted in order of
    increasing free times)
16. Add new task on task list (sorted in order of increasing
    start times)
```

Figure 4.2: Our Intelligent Stuffing algorithm

When a new task $T1$ arrives, the algorithm searches the free space list SL and places it on the leftmost edge of FS_1 (according to its alignment status). This action reduces the size of FS_1 as shown in the middle of Figure 4.2(a), toggles the alignment status of FS_1 from leftmost to rightmost, and creates a new free space FS_2. FS_2 has (CL_2, CR_2) dimension and its free time is FT_2 and leftmost alignment status.

Assume there is another task $T2$ simultaneously arriving with $T1$ the free space list SL will be processed again. Because the alignment status of FS_1 was changed to rightmost, $T2$ will be placed on rightmost edge of FS_1. This action reduces the FS_1 size as shown in Figure 4.2(a) (bottom) and again toggles the alignment status of FS_1 to leftmost. The size of FS_2 is also adjusted and a new free space FS_3 (CL_3,CR_3) is created with free time FT_3 and leftmost alignment status. By keeping tasks $T1$ and $T2$ on the edges, the largest space possible is created, so future tasks can be scheduled earlier and we can address the problem of previous algorithms for both case 1 and case 2 as shown in Figure 4.2(b).

There are two operating modes: *speed* and *quality*. In the speed mode, the algorithm execution time is more important than the quality of scheduling and placement. While the quality mode is designed for higher utilization of the resources. The pseudocode of our algorithm is presented in Figure 4.2(c). When a new task arrives, our algorithm walks through the SL to find a first fit free space avoiding conflicts with the scheduled tasks in the TL (line 1 to 11). The first fit free space has the earliest free time which enough columns of reconfigurable units to fit the task.

If quality mode is chosen, lines 6 to 10 are executed for better quality (in speed mode those lines are skipped to reduce the algorithm execution time). In lines 6 to 8, a placement of the task at the opposite position to the alignment status is attempted. This action increases the quality, but it requires additional algorithm time. If the task still conflicts with the currently scheduled tasks in the TL (line 9), the alignment status of the corresponding free space is set to its initial condition (line 10).

In line 12, the first fit free space without conflicts with the TL list is found, however this space may be wider than that the task requirements. The task is placed on the FS_i edge according to its alignment status. As mentioned earlier, every placement changes the size and toggles the alignment status of the used free space (line 13). This action can also affect the other free space sizes (line 14) and adds a new free space in the SL (line 15) in addition to the new scheduled task in the TL (line 16).

4.1.4 Evaluation

Experimental Setup

We implemented four different algorithms (the Stuffing [46] [47] (STF), the Classified Stuffing [48] (CTF) and our algorithm using *speed* mode (ISS) and *quality* mode (ISQ)) in ANSI-C and run them on a Pentium-IV 3.4 GHz PC using the same task sets. The simulated device consists of 96 columns to model Xilinx XCV1000 (96x64 reconfigurable units). The task widths and execution times of tasks are generated randomly in [1,96] columns of reconfigurable units and [1,1000] time units. We generate randomly 20 tasks for each task set and run all algorithms using 100,000 task sets. The evaluation is based on four performance parameters: total wasted area (TWA), total schedule time (TST), total response time (TRT), and total algorithm execution time (TAT) (μs).

Experimental Results

Table 4.1 shows that even in *speed* mode our algorithm utilizes the FPGA better, decreasing the wasted area compared to the Stuffing by 64.5 %. In addition, it makes the overall application execution 1.1 % faster and has 17.4 % shorter waiting time. The *speed* mode is not only faster than the Classified Stuffing (5 % shorter algorithm execution time) but also utilizes the FPGA more effective by decreasing the wasted area by 53 %. Furthermore the application execution is reduced by 0.7 % with 12.8 % shorter total waiting time.

Table 4.1: Obtained results using 100,000 task sets (TWA:total wasted area, TST:total schedule time, TRT:total response time and TAT:total algorithm execution time)

Performance parameters	STF	CTF	ISS	ISQ
TWA(space-time units)	1035449499	783069435	367934139	106709691
TST(time units)	651128773	648499814	644175488	641454400
TRT(time units)	335229077	317655028	276949454	230250447
TAT(μs)	2076694	2184614	2074848	2168651

In *quality* mode the wasted area is decreased by 89.7 % compared to the Stuffing with only 4.2 % algorithm execution time overhead (saving the alignment status bit and finding alternative placements). Moreover it makes the application running 1.5 % faster with 31.3 % shorter total waiting time. In respect to the Classified Stuffing the *quality* mode is not only faster by 0.7 % in terms of algorithm execution time but also decreases the FPGA wasted area by 86.4 %. Additionally, the overall application execution time is reduced by 1.1 % with

27.5 % better total waiting time.

4.2 Proposed 3D Compaction Algorithm for 2D Area Model

4.2.1 Problem of Scheduling and Placement on 2D Area Model

Given a task set representing a multitasking application with their arrival times a_i, life-times lt_i, widths w_i and heights h_i, online task scheduling and placement algorithms targeting the 2D area model of partially reconfigurable devices have to determine placements and starting times for the task set such that there are no overlaps in space and time among all tasks. The goals of the algorithms are: a) to utilize effectively the available FPGA resources (minimize wasted volume); b) to accelerate the overall application on the FPGA (minimize schedule time); c) to start executing arriving tasks on the FPGA earlier (minimize waiting time) and d) to keep the runtime overhead low (minimize the algorithm execution time).

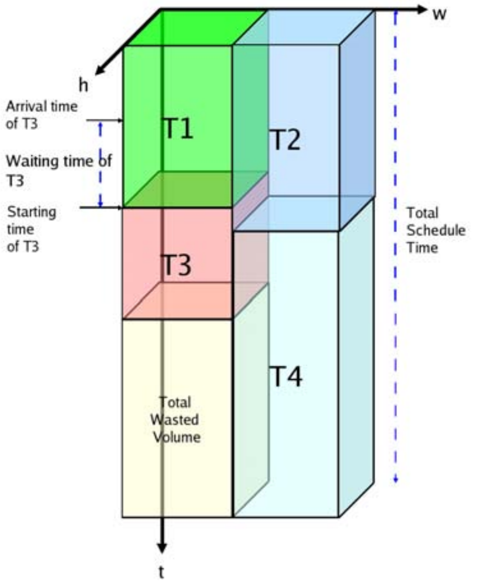


Figure 4.3: Problem of scheduling and placement on 2D area model

We define the total wasted volume as the overall number of area-time units that are not utilized as illustrated in Figure 4.3. Total schedule time is the total number of time units for the execution of all tasks. Waiting time is the difference between task starting and arrival times (in time units). The algorithm execution time is the time needed to schedule and place the arriving task.

4.2.2 Blocking-Aware Algorithm Main Idea

Blocking-unaware algorithms do not consider whether future incoming tasks will be blocked while deciding on the current task placement position. This can be seen as if drivers parking their vehicles at completely random places and hence preventing other drivers of parking their cars. Figure 4.4 (left) illustrates the behavior of online scheduling and placement algorithms that do not have blocking-awareness. In this simple example, task T3 is becoming an obstacle for task T4 arriving after T3.

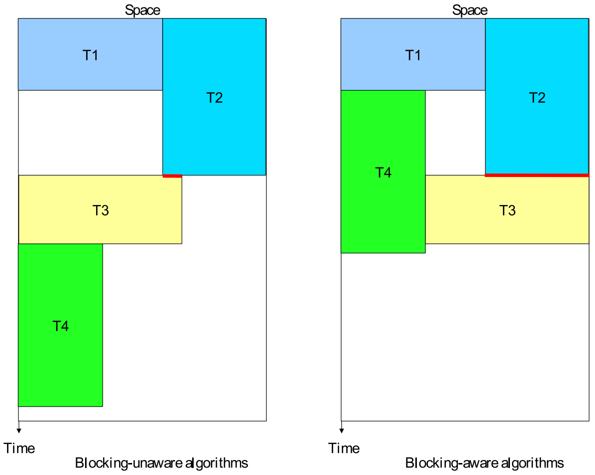


Figure 4.4: Basic idea of blocking-aware algorithm

To tackle this problem, we introduce an algorithm that can avoid placement decisions that will become an obstacle for future HW tasks. By placing task T3 to a different location as shown in Figure 4.4 (right), the proposed algorithm can avoid task T3 to be an encumbrance for task T4 that can be started earlier now. By this early scheduling T4 can finish its execution faster. To give the algorithm the necessary knowledge to avoid such ”blocking-effect”, it places tasks at locations as much as possible touching all prior tasks illustrated with bold lines on the figure. In the next section, we will provide a more detail explanation of this heuristic, termed 3D total contiguous surface (3DTCS).

4.2.3 3D Total Contiguous Surface (3DTCS) Heuristic

A hardware task on a 2D partially reconfigurable device using 2D area model can be illustrated as a 3D box. The first two dimensions are the required area `(wh)` on the device for running the task. The other dimension is the time dimension `(t)`. To pack hardware tasks compactly during run time at the earliest time, we propose a new heuristic, named 3D total contiguous surface (3DTCS) heuristic.

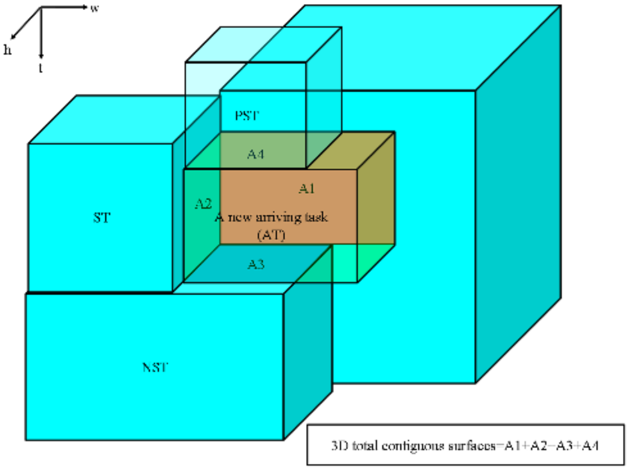


Figure 4.5: 3D total contiguous surface (3DTCS) heuristic

The 3DTCS is the sum of all surfaces of an arriving task that is contacted with the surfaces of other scheduled tasks as depicted in Figure 4.5. The 3DTCS contains two components:

- the horizontal contiguous surfaces with previous scheduled tasks and next scheduled tasks;
- the vertical contiguous surfaces with scheduled tasks and the FPGA boundary.

In a simple example depicted in Figure 4.5, the horizontal contiguous surfaces with a previous scheduled task (PST) A4 and with a next scheduled task (NST) A3 in the figure give this heuristic an awareness on avoiding ”blocking-effect”; while the other surfaces A1 and A2 (vertical contiguous surfaces) give this heuristic to better pack tasks in time and space. As a result, the proposed algorithm has a full 3D-view of the positions of all scheduled and placed tasks.

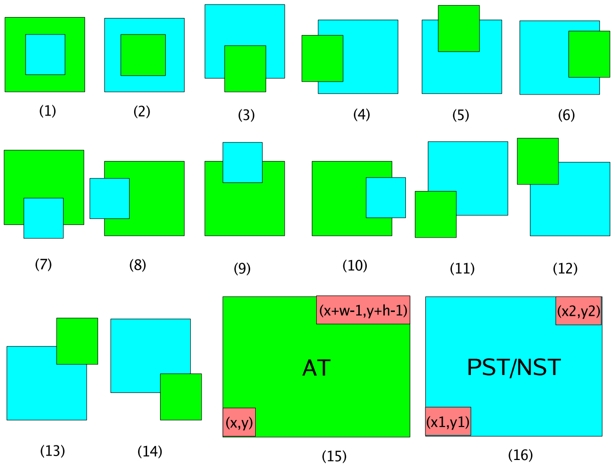


Figure 4.6: Horizontal contiguous surfaces

Intuitively, a higher 3DTCS value will result in more compaction both in space and time. This 3DTCS heuristic gives our proposed 3D compaction algorithm with blocking-aware ability to pack tasks better as it has a more complete view of all dimensions.

Figure 4.6(1)-(14) and Table 4.2 show all the placement positions and their corresponding computations of horizontal contiguous surfaces. The arriving task (AT), with width w and height h, has a bottom-left coordinate (x, y) as shown in Figure 4.6(15). The arriving task can be contacted with the previous scheduled task (PST) and (or) the next scheduled task (NST) to produce the horizontal contiguous surfaces. The scheduled task has a bottom-left coordinate (x_1, y_1) and a top-right coordinate (x_2, y_2) as illustrated in Figure 4.6(16).

The arriving task can be contacted with scheduled tasks and (or) FPGA boundary to produce the vertical contiguous surfaces. All placement positions of the arriving task (AT) and their corresponding computations of vertical contiguous surfaces with the scheduled task (ST) are shown in Figure 4.7(a) and Table 4.3. The arriving task with a life-time lt is started execution at time t_s; the finishing time of scheduled task is denoted as t_f. Computations of vertical contiguous surfaces between the arriving task with the FPGA boundary are illustrated in Figure 4.7(b) and formulated in Table 4.4.

Table 4.2: Computations of horizontal contiguous surfaces for positions in Figure 4.6(1)-(14)

Positions	Horizontal contiguous surfaces
(1)	$(x_2 - x_1 + 1)(y_2 - y_1 + 1)$
(2)	wh
(3)	$w(y + h - y_1)$
(4)	$(x + w - x_1)h$
(5)	$w(y_2 - y + 1)$
(6)	$(x_2 - x + 1)h$
(7)	$(x_2 - x_1 + 1)(y_2 - y + 1)$
(8)	$(x_2 - x + 1)(y_2 - y_1 + 1)$
(9)	$(x_2 - x_1 + 1)(y + h - y_1)$
(10)	$(x + w - x_1)(y_2 - y_1 + 1)$
(11)	$(x + w - x_1)(y + h - y_1)$
(12)	$(x + w - x_1)(y_2 - y + 1)$
(13)	$(x_2 - x + 1)(y_2 - y + 1)$
(14)	$(x_2 - x + 1)(y + h - y_1)$

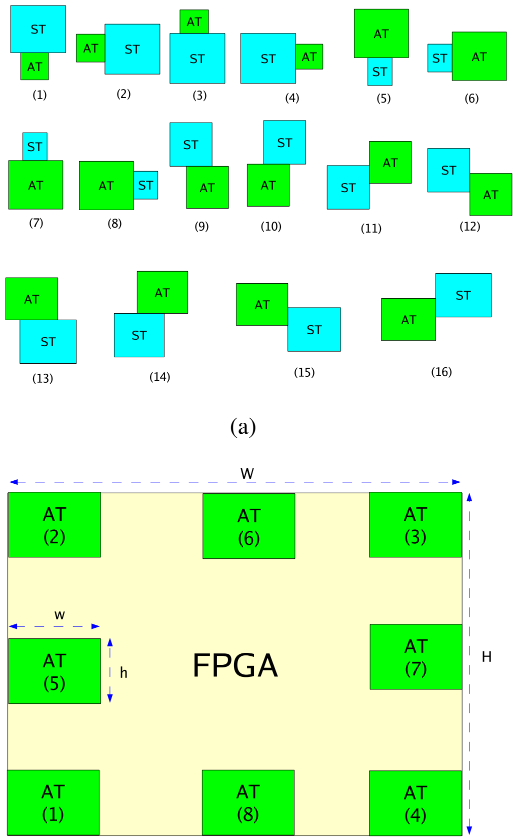
Table 4.3: Computations of vertical contiguous surfaces with scheduled tasks for positions in Figure 4.7(a)(1)-(16)

Positions	Vertical contiguous surfaces with scheduled tasks
(1),(3)	$w.min(lt, (t_f - t_s))$
(2),(4)	$h.min(lt, (t_f - t_s))$
(5),(7)	$(x_2 - x_1 + 1).min(lt, (t_f - t_s))$
(6),(8)	$(y_2 - y_1 + 1).min(lt, (t_f - t_s))$
(9),(14)	$(x_2 - x + 1).min(lt, (t_f - t_s))$
(10),(13)	$(x + w - x_1).min(lt, (t_f - t_s))$
(11),(15)	$(y_2 - y + 1).min(lt, (t_f - t_s))$
(12),(16)	$(y + h - y_1).min(lt, (t_f - t_s))$

4.2.4 The 3D Compaction (3DC) Algorithm

Figure 4.8 shows the pseudocode for the proposed 3D Compaction (3DC). The algorithm maintains two linked lists: the execution list and the reservation list. The execution list saves the information of all currently running tasks sorted in order of increasing finishing times; the reservation list contains the information of all scheduled tasks sorted in order of increasing starting times. The information stored in the lists are the bottom-left coordinate (x_1, y_1), the top-right coordinate (x_2, y_2), the starting time t_s, the finishing time t_f, the task name, the next pointer, and the previous pointer.

In lines 1-13, the algorithm computes the starting time matrix (STM) with respect to the arriving task area wh on the device area WH. The algorithm collects all possible positions that have enough space for the arriving task by



(a)

(b)

Figure 4.7: Vertical contiguous surfaces with scheduled tasks (a) and the FPGA boundary (b)

scanning the executing and reservation lists. The algorithm fills each element of the STM with the arrival time of incoming task `a` (lines 1-3). The algorithm updates groups of elements that are affected by all executing tasks in execution list (lines 4-8) and by all scheduled tasks in reservation list (lines 9-13).

In line 14, the algorithm collects all the best positions (candidates) that have the earliest starting time (the best starting time positions: the best positions in terms of starting time) from the STM.

Since the algorithm not only wants to get the best position in terms of starting time (time domain) but the best position in terms of space (space domain) as well. To pack compactly tasks, we propose to use the 3DTCS heuristic as

Table 4.4: Computations of vertical contiguous surfaces with the FPGA boundary for positions in Figure 4.7(b)(1)-(8)

Positions	Vertical contiguous surfaces with the FPGA boundary
(1)-(4)	$(w+h)lt$
(5),(7)	$h.lt$
(6),(8)	$w.lt$

presented earlier. The algorithm computes the 3DTCS (line 16) using formulas from Table 4.2 to Table 4.4 and chooses the best position from all the best starting time positions. Hence, the algorithm does not need to compute the 3DTCS for all positions; it only computes the 3DTCS for the best positions (candidates) (line 15). Intuitively, the highest 3DTCS value gives the best position in terms of packing to avoid ”blocking-effect”.

Besides the 3DTCS heuristic, the algorithm also uses the sum of finishing time difference (SFTD) heuristic for all scheduled tasks that vertically contacted with the arriving task (referred as a VC set). The algorithm computes current SFTD ($c\_SFTD = \sum_{\forall tasks \in VC} |t_s + lt - t_f|$) in line 17. The SFTD heuristic gives our algorithm an ability to group tasks with similar finishing times to get large free space during deallocations.

The algorithm chooses the position with the highest 3DTCS value and the lowest SFTD value for allocating the arriving task (lines 18-27). Allocating the arriving tasks at the highest 3DTCS compacts the tasks both in time and space; while grouping tasks with similar finishing times creates more possibility to produce larger free space during deallocations.

The algorithm allocates the incoming task when there is available space for the task at its arrival time; otherwise, the algorithm needs to schedule the task for future execution. If the arriving task can be allocated at its arrival time (line 28), it will be executed immediately and added in the execution list (line 29); otherwise, it is inserted in the reservation list (line 30).

When the tasks in the reservation list are executed, they are removed from the reservation list and added in the execution list. The finished tasks in the execution list are deleted after execution. These updating processes are executed when the lists are not empty (lines 31-34).

The time complexity analysis of 3DC is presented in Table 4.5. In which W, H, N_{ET}, N_{RT} are the FPGA width and height, the number of executing tasks in the execution list and the number of reserved tasks in the reservation list.

```
1. for (y=1;y<=H-h+1;y++)
   {
2.       for (x=1;x<=W-w+1;x++)
         {
3.               STM(x,y)=a
         }
   }

4. for all tasks in execution list
   {
5.       for (y=max(1,$y_1$-h+1);y<=min($y_2$,H-h+1);y++)
         {
6.               for (x=max(1,$x_1$-w+1);x<=min($x_2$,W-w+1);x++)
                 {
7.                       if (STM(x,y) < $t_f$)
                         {
8.                               STM(x,y)=$t_f$
                         }
                 }
         }
   }

9. for all tasks in reservation list
   {
10.      for (y=max(1,$y_1$-h+1);y<=min($y_2$,H-h+1);y++)
         {
11.              for (x=max(1,$x_1$-w+1);x<=min($x_2$,W-w+1);x++)
                 {
12.                      if ((STM(x,y) < $t_f$) AND (STM(x,y)+lt>$t_s$))
                         {
13.                              STM(x,y)=$t_f$
                         }
                 }
         }
   }

14. collect all positions from STM that have the earliest starting time
15. for all above positions
   {
16.      c_3DTCS=compute 3D contact surfaces
17.      c_SFTD=compute sum of finishing time difference
18.      if (c_3DTCS>3DTCS_max AND c_SFTD<SFTD_min)
         {
19.              best_position=current position
20.              3DTCS_max=c_3DTCS
21.              SFTD_min=c_SFTD
         }
22.      else if (c_3DTCS>3DTCS_max)
         {
23.              best_position=current position
24.              3DTCS_max=c_3DTCS
         }
25.      else if (c_3DTCS=3DTCS_max AND c_SFTD<SFTD_min)
         {
26.              best_position=current position
27.              SFTD_min=c_SFTD
         }
   }
28. if best_starting_time=arrival time
   {
29.      add task to the execution list
   }
   else
   {
30.      add task to the reservation list
   }
31. if  the reservation list is not empty
   {
32.      update reservation list
   }
33. if  the execution list is not empty
   {
34.      update execution list
   }
```

Figure 4.8: Pseudocode of 3D Compaction algorithm

The main difference between our algorithm and existing algorithms is the presence of the 3D compaction ability. Because of this 3D compaction ability, our algorithm can avoid ”blocking-effect”. In contrast, the existing algorithms do not have the blocking-awareness. Some existing algorithms only have the 2D compaction ability; instead, our algorithm has the 3D compaction ability to compact tasks both in time and space domains. Besides, the algorithm also has an ability to group tasks with similar finishing times to achieve larger free space during deallocations. In the CR, every element of their EA matrix is checked to know if it falls into the coverage rectangles of execution and scheduling tasks for updating as shown in [52]. In contrast, our algorithm updates the STM matrix in groups of elements affected by all executing (lines 5-6) and scheduled tasks (lines 10-11); the algorithm does not need to check each element for updating. As a result, our algorithm computes starting times faster than CR. Moreover, 3DC does not need to compute boundary values for all reconfigurable units of its free space in the periphery reducing the runtime overhead compared to CR as will be presented later.

Table 4.5: Time complexity analysis of 3D Compaction algorithm

Lines	Time Complexity
1-3	$O(W * H)$
4-8	$O(W * H * N_{ET})$
9-13	$O(W * H * N_{RT})$
14	$O(W * H)$
15-27	$O(W * H * max(N_{ET}, N_{RT}))$
28-30	$O(max(N_{ET}, N_{RT}))$
31-32	$O(N_{RT})$
33-34	$O(N_{ET})$
Total	$O(W * H * max(N_{ET}, N_{RT}))$

4.2.5 Evaluation

Experimental Setup

We have built a discrete-time simulation framework in C to evaluate the proposed algorithm. The framework was compiled and run under Linux operating system on a Pentium-IV 3.4 GHz PC. To better evaluate the algorithm with synthetic workloads, (1) we modeled realistic random hardware tasks to be executed on a realistic target device; (2) we evaluated the algorithm not only in terms of scheduling and placement quality but also in terms of runtime overhead. Since the algorithms are online, the information of arriving tasks is unknown until their arrival times. We model a realistic FPGA with 116 columns and 192 rows of reconfigurable units (Virtex-4 XC4VLX200).

Scheduling and Placement Quality using Synthetic Workloads

To model realistically the synthetic hardware tasks, we use the same realistic hardware tasks from Chapter 3 to obtain the information of hardware task size range as a reference for our random task set generator. The task widths and heights are randomly generated in the range [7..45] reconfigurable units to model hardware tasks between 49 and 2025 reconfigurable units to mimic the results of synthesized hardware units. Every task set consists of 1000 tasks, each of which has a life-time and task size. The life-times are randomly generated in [5..100] time units, while the intertask-arrival periods are randomly chosen between one time unit and a specified maximum intertask-arrival period. Total tasks per arrival are randomly generated in [1..15].

Our 3DC is designed for 2D area model. Therefore for fair comparison, we only compare our algorithm with algorithms that support 2D area model. Since the RPR [50], the Classified Stuffing [48], the Intelligent Stuffing were designed only for 1D area model, we do not compare them with our 3DC.

Since the Stuffing outperforms the Horizon as presented in [46], we do not compare our algorithm to the Horizon. In [52], the CR outperforms the original 2D Stuffing [46] [47] and the 2D Window-based Stuffing [51]; therefore, we only compare our algorithm to the CR.

To evaluate the 3DC, we have implemented three different algorithms: the CR [52] using BL (Bottom-Left) scheme (CR_BL), the CR [52] using BV (Boundary Value) scheme [53] (CR_BV), and our 3DC. The evaluation is based on three performance parameters: total schedule time, waiting time, and total wasted volume.

The CR does not have a blocking-awareness. Instead, our algorithm uses a 3D compaction for avoiding ”blocking-effect”. As a consequence, our algorithm has a better quality than the CR. The 3DC has up to 4.8 % shorter schedule time, 38.4 % shorter waiting time, and 22.9 % less wasted volume compared to the CR as shown in Figure 4.9.

The system idle time increases when the maximum inter-task arrival period increases; as a result, the average total schedule time and the average wasted volume increase.

The system is busier when the maximum inter-task arrival period decreases; tasks arrive more frequently to the system. Hence, it is more difficult to schedule tasks. Consequently, the average waiting time increases.

Scheduling and Placement Quality using Real Workloads

To evaluate the 3DC with real workloads, the same realistic hardware tasks from Chapter 3 are used. In the simulation, we assume that the life-time $\mathtt{lt}_i$ is the sum of reconfiguration time $\mathtt{rt}_i$ and execution time $\mathtt{et}_i$. The experimental results with real workloads are presented in Figure 4.10.

Figure 4.10 shows that the superiority of our algorithm is not only applicable for synthetic tasks but also for real tasks. Evaluation with real tasks shows that our algorithm has up to 4.6 % shorter schedule time, 75.1 % shorter waiting time, and 9.9 % less wasted volume compared to the CR.

Algorithm Execution Time Results

To complete the evaluation, we also study the algorithm execution time since the execution time of online task scheduling and placement is considered as an overhead for the overall execution time of the applications. To show the effect

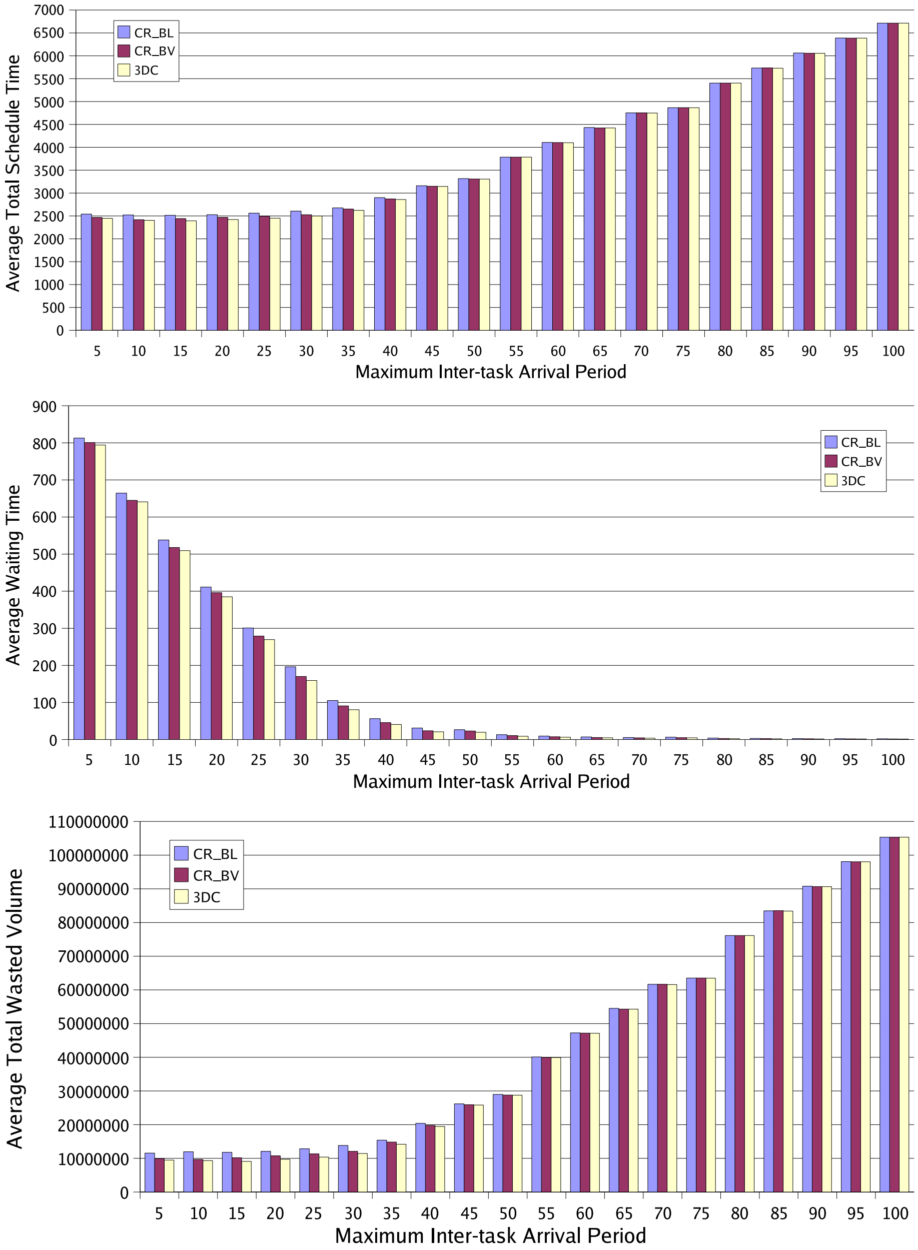


Figure 4.9: Evaluation with synthetic workloads

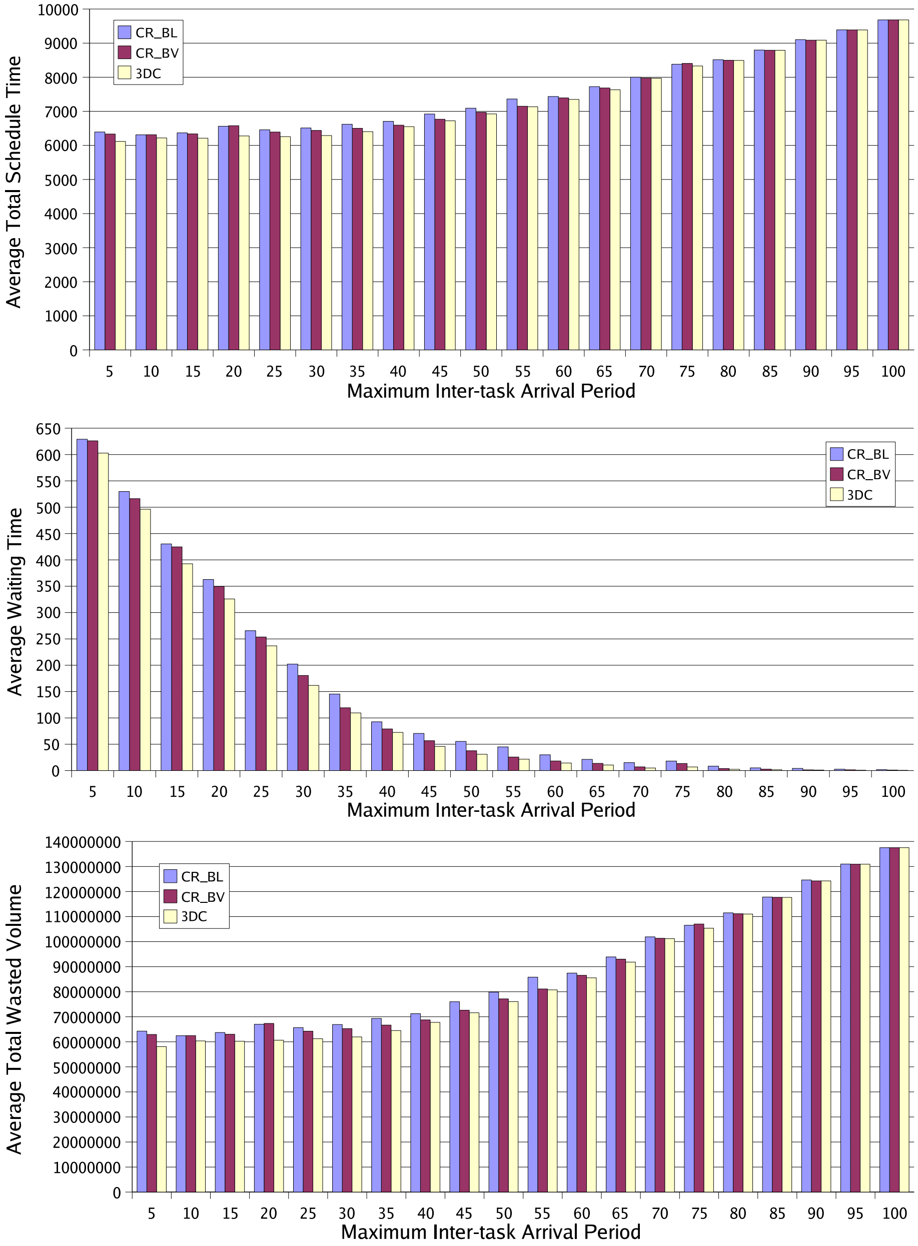


Figure 4.10: Evaluation with real workloads

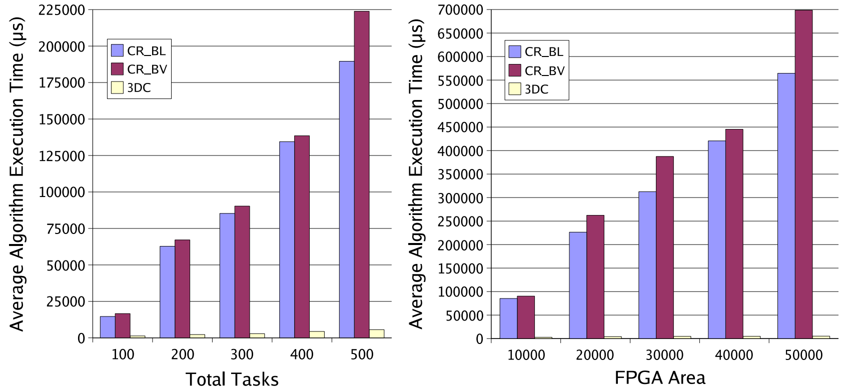


Figure 4.11: Evaluation of algorithm execution time

of total number of scheduled and running tasks as well as FPGA area, we do simulation by changing these parameters as presented in Figure 4.11.

Figure 4.11 shows that our 3DC runs up to 133 times faster than the CR. The speed up will be higher for more scheduled and running tasks as well as for larger FPGA fabrics. Since the CR uses the boundary value heuristic for searching placement, the CR needs to compute boundary values for all reconfigurable units of its free space in the periphery. In contrast, our 3DC computes the 3DTCS only in one step. Moreover, the updating is done per each element of the matrix in the CR; each element is needed to be checked with all executing tasks and scheduled tasks. In contrast, our algorithm updates the matrix in groups of elements located by all executing tasks and scheduled tasks; the algorithm does not need to check each element for updating. As a result, our 3DC has less runtime overhead than the CR by avoiding the CR's long boundary value computation and speeding up the starting times computation.

More FPGA area creates additional area suitable for the arriving task (more free volume) and more total number of scheduled and running tasks forces algorithms to check more tasks; as a result, the algorithms need more time to compute the matrix for finding starting time (all algorithms), all boundary values for all more candidates (CR algorithm) and all 3DTCS for all more candidates (3DC algorithm). Because of its long boundary value and matrix computations, the CR execution time increases faster than our 3DC.

4.3 Summary

In this chapter, we proposed two novel online HW task scheduling and placement algorithms. The Intelligent Stuffing algorithm, designed for 1D area model, and the 3D Compaction (3DC), aiming at the 2D area model.

The main difference between our Intelligent Stuffing algorithm and related art is the additional alignment status of the free space segments and its handling. This status allows our algorithm to maximize the free space ares making task placement position decisions and allow earlier placing of further tasks. Moreover, the SUR computation is not needed making it faster than the Classified Stuffing. Experimental results show that our Intelligent Stuffing outperforms existing algorithms in terms of reduced total wasted area up to 89.7%, has 1.5 % shorter schedule time and 31.3% faster response time.

To avoid ”blocking-effect” we proposed a new 3DTCS heuristic in a novel blocking-aware algorithm, 3D Compaction (3DC). The 3DC can place and schedule tasks more compactly and is able to group similar finishing time tasks to form larger free area. Since state of the art uses the boundary value heuristic for searching suitable placement, it needs to compute the values for all reconfigurable units of its free space in the periphery. In contrast, our 3DC computes the 3DTCS in a single step. In addition, the updating is done per each element of the matrix for finding starting time in the previous algorithm; each element is checked with all executing and scheduled tasks. Our 3DC updates the matrix in groups of elements located by all executing and scheduled tasks. The experimental results show that the 3DC not only has better scheduling and placement quality (up to 4.8 % shorter schedule time, 75.1 % shorter waiting time, and 22.9 % less wasted volume) but also has lower runtime overhead compared to existing algorithms.

Note. The content of this chapter is based on the following papers:

T. Marconi, Y. Lu, K.L.M. Bertels, G. N. Gaydadjiev, **Online Hardware Task Scheduling and Placement Algorithm on Partially Reconfigurable Devices**, Proceedings of International Workshop on Applied Reconfigurable Computing (ARC), March 2008.

T. Marconi, Y. Lu, K.L.M. Bertels, G. N. Gaydadjiev, **3D Compaction: a Novel Blocking-aware Algorithm for Online Hardware Task Scheduling and Placement on 2D Partially Reconfigurable Devices**, Proceedings of the International Symposium on Applied Reconfigurable Computing (ARC), March 2010.

5

Low Power Logic Element for FPDs

Although various techniques have been proposed for power reduction in field-programmable devices (FPDs), they are all based on conventional logic elements (LEs). In the conventional LE, the output of the combinational logic (e.g., the lookup table (LUT) in many PLDs and FPGAs) is connected to the input of the storage element; while the D flip-flop (DFF) is always clocked even when not necessary. Such unnecessary transitions waste power. To address this problem, we propose a novel low power LE with reduced number of transitions. The differences between our LE and the conventional LE are in the flip-flops type used and the internal LE organization. Instead of using DFFs, we use T flip-flops with the T input permanently connected to logic value one. Instead of connecting the output of the combinational logic to the FF input, we use it as the FF clock. The proposed LE is evaluated using transistor-level circuit simulation in terms of power consumption, performance, and area using the MCNC benchmark circuits. Besides, we also evaluate our proposal using a real CAD tool and a real FPGA by forcing the existing tool to implement circuits behaving like our proposed LE.

This chapter is organized as follows. The problem of FPD high power consumption for runtime reconfigurable systems and our proposal are emphasized in Section 5.1. In Section 5.2, we propose our low power LE to reduce power consumption. Our proposal is evaluated against state of the art in Section 5.3. Finally, Section 5.5 ends with the conclusions.

5.1 Introduction

Many techniques have been proposed for power reduction in FPDs. However, all existing power reduction techniques target what we call a ”conventional logic element”. This conventional logic element (LE) has been used by researchers of FPDs since it was patented by Birkaner and Chua in 1978 [57]. Although FPDs have been improved significantly since the original proposal, they still make use of a proposal dated 1978 that may need to be reconsidered. The conventional LE contains the combinational logic (e.g., the lookup table LUT in FPGAs) and the storage element (D flip-flop). The output of the combinational logic is connected to the input of the storage element; the clock input of D flip-flop (DFF) is connected to the clock signal. Since the DFF clock input is connected directly to the clock signal, the DFF is always clocked even when this is not needed. For example, when $D = Q$, the DFF does not need to be clocked. Such unnecessary transitions waste power in FPDs using the conventional LEs. This is related to the fact that even low-power flip-flops consume power during logic transition from zero-to-zero and from one-to-one as shown in [201].

To solve this problem, we propose a novel LE for reduced FPDs power consumption. The proposed LE can be used in any kinds of FPDs: Simple PLDs (SPLDs), Complex PLDs (CPLDs) as well as Field-Programmable Gate Arrays (FPGAs). The differences between our LE and the conventional proposal are in the flip-flops type and the LE internal organization. Instead of using D flip-flops, we use T flip-flops with T input permanently at logic one (T=1). This is related to the fact that designing sequential circuits using TFFs is more power efficient than DFFs as reported in [200]. The output of the combinational logic in our case is connected to the clock input of the FF. As a result, our LE is able to block unnecessary clock transitions without using additional clock gating logic. Since unnecessary clock transitions are avoided, the clock power is reduced. By avoiding unnecessary clock transitions, the overall switching activity inside the LEs is also reduced. As a result, FPDs using the proposed LEs consume also less logic power (total power inside LEs) compared to FPDs using conventional LEs. Because of the reduced activity, the interconnect activity among LEs is also reduced. Our approach does not require additional controller for gating clock activity and will potentially also save power and area in respect to this.

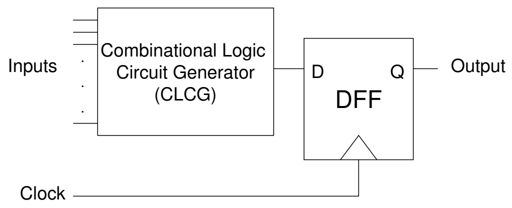
In conventional LEs, the FF is clocked when the D input has a stable logic value provided by the combinational logic and determined by the FF setup time. In our LEs, since the T input of the FF is always in logic one, the FF is always ready to be clocked. As a consequence, logic circuits implemented using our LEs can be clocked faster than conventional LEs.

The Microelectronic Center of North Carolina (MCNC) benchmark circuits [58] are used to evaluate both FPD types in 45 nm BSIM4 CMOS technology

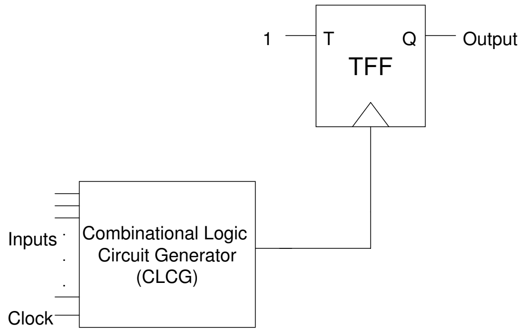
[60]. We use LTSPICE tools [59] for transistor-level circuit simulations with nominal supply voltage VDD of 1.2 V. The evaluation is performed in terms of total power, logic power, clock power, interconnect power, dynamic power, static power, speed, and LE area.

5.2 The Proposed Logic Element

The purpose of LEs in FPDs is to provide the basic programmable combinational logic and storage elements used in digital systems. An LE contains a combinational logic circuit generator (CLCG) and a storage element as shown in Figure 5.1. The CLCG is used for the combinational function, while the storage element is used for storing temporary results.



(a) Conventional logic element



(b) Our logic element

Figure 5.1: Logic elements

In conventional LEs, the output of CLCG is connected to the input of the storage element as illustrated in Figure 5.1(a). The storage element in the conventional LE is a D Flip-flop (DFF). Since the clock input of DFF is connected to

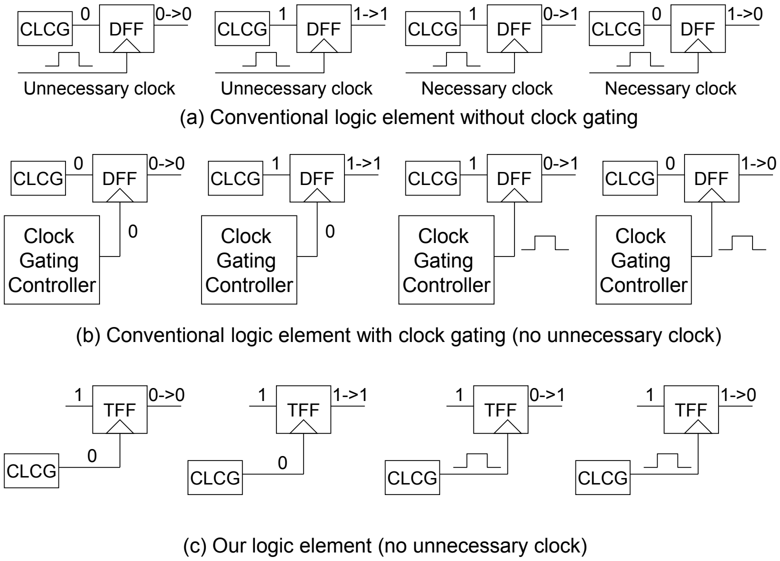

CLCG
DFF
TFF
0
1
0->0
1->1
0->1
1->0
Unnecessary clock
Necessary clock
Clock Gating Controller
(a) Conventional logic element without clock gating
(b) Conventional logic element with clock gating (no unnecessary clock)
(c) Our logic element (no unnecessary clock)

Figure 5.2: Basic operations of logic elements

the clock signal, the DFF is always clocked. When the D input of DFF has a different value compared to its output Q ($D \neq Q$), the DFF needs to be clocked in order to update its state as presented in Figure 5.2(a). Otherwise, when $D = Q$, the DFF does not need to be clocked. Such unnecessary transitions will waste power in the conventional LEs.

To stop unnecessary clock transitions in conventional LEs, clock gating was introduced in previous work [74], [75], and [76]. In clock gating, the clock input of DFF is not anymore connected directly to the clock signal, but it is controlled by the clock gating controller as shown in Figure 5.2(b). The clock gating controller blocks the clock signal for reaching DFFs clock inputs when the DFFs should not be clocked ($D = Q$). As a result, the unnecessary clock transitions can be avoided for power saving. The drawback of clock gating is the need of additional controllers that consume additional area and power. To reduce this overhead, the controller usually does not control an individual FF, but it controls a group of FFs together. As a result, the clock gating cannot block all of the unnecessary clock transitions.

To solve the above issues of conventional LEs, we propose a novel low power LE depicted in Figure 5.1(b). The differences between our LE and the conventional LE are in the type of FFs and the LE organization. Instead of using

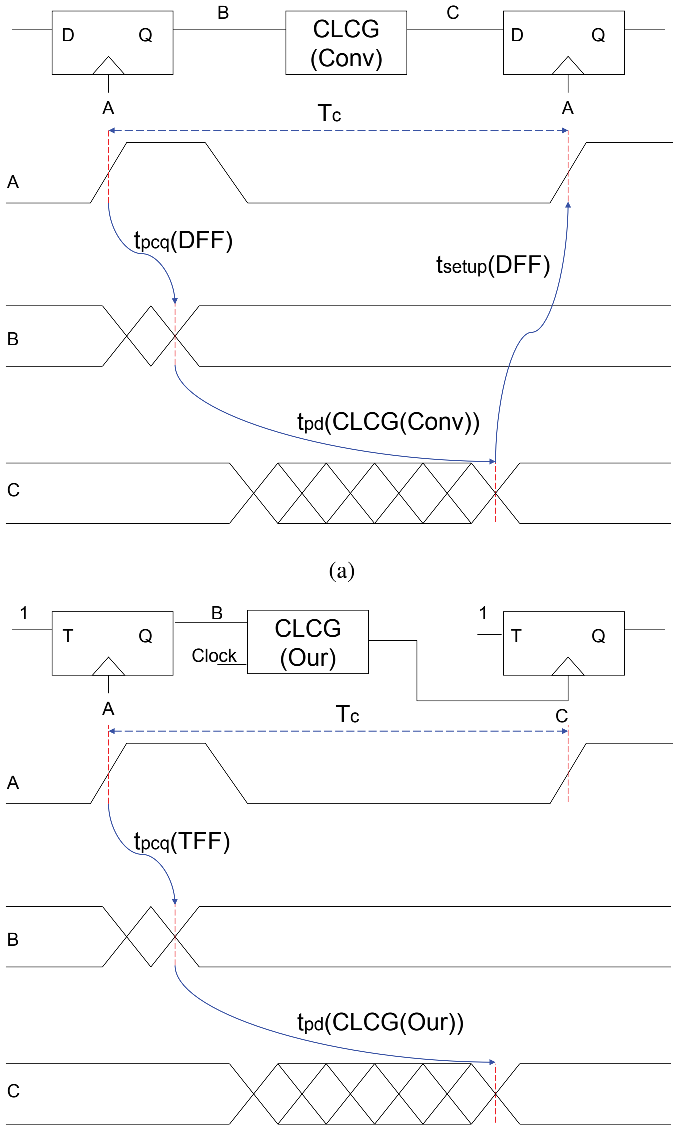
DFFs, we use T flip-flops (TFFs) with the T input kept at logic one. The output of the CLCG is connected to the FF clock input. No clock signal is directly connected to the TFF; the clock signal is connected to the TFF through the CLCG when required. In FPGAs, CLCGs are implemented using LUTs. In the case that one of the inputs of the LUT is used for feeding the clock signal, the LUT capacity is effectively decreased. This will not be a problem, since not all inputs of LUTs are used in real FPGA designs as reported in [21], we can use these unused inputs for free to feed the clock signal.

The benefits of our LE are avoiding unnecessary clock transitions while omitting the additional clock gating controller as shown in Figure 5.2(c). The CLCG avoids clock transitions to be propagated to an individual FF when the state of the FF will not change. As a result, the unnecessary clock transitions are totally avoided at the level of individual FFs and hence dynamic power is reduced. Additional power and area are also saved in comparison to the clock gating approach, since the additional controller is not present.

Although not shown for simplicity in Figure 5.2, the present state and inputs are used to generate the next state function in the conventional LE; while in our circuit, the present state, inputs and clock signal are used to generate function to control TFFs clocks. An a result, the way we design logic circuit will be different compared to the conventional approach. In conventional circuits, the data path, the control path, and the clock are separated. In our circuits, all these paths are combined together into a single unified path.

Allowing faster clock rates than the conventional LEs is one additional advantage of our proposal. The FF can be clocked properly if its input is stable at least before its setup time. In conventional LE, the input value of the DFF is not constant; it depends on the output of the connected CLCG. In our LE, since the T input of the TFF is constant ($T = 1$), the TFF is always ready to be clocked. As a consequence, logic circuits implemented using our LEs can be clocked faster than logic circuits using conventional LEs.

The shortest possible clock timing diagrams for circuits using our LEs compared to the conventional LEs are presented in Figure 5.3. Please note that this experiment is used to investigate the differences in maximal clock rates and is in disadvantage for our proposal since the longest propagation delay of the first level DFF in Figure 5.3(a) is $t_{pdq}(DFF)$ (the Data-to-Q propagation delay). The CLCG (Our) in Figure 5.3(b) has the clock as an additional input. This, however, does not impact the first-to-second stage shortest possible clock timing due to the $t_{pcq}(TFF)$ delay that has to be satisfied. Please also note that the clock signal of the first level TFF (A) is pro-



(a)

(b)

Figure 5.3: Shortest clock timing of conventional (a) and our (b) logic elements

duced by the previous level CLCG (Our) not shown on the figure for simplicity. In the figure, $t_{pcq}(DFF)$ is the clock-to-Q propagation delay of DFF; $t_{pd}(CLCG(Conv))$ is the propagation delay of conventional CLCG; $t_{pd}(CLCG(Our))$ is the propagation delay of our CLCG; $t_{setup}(DFF)$ is the setup time of DFF; $t_{pcq}(TFF)$ is the clock-to-Q propagation delay of TFF. From this figure, we can obtain the clock period of the circuit using conventional LEs as $T_c(Conv) \geq t_{pcq}(DFF) + t_{pd}(CLCG(Conv)) + t_{setup}(DFF)$ (1) and the clock period of the circuit using our LEs as $T_c(Our) \geq t_{pcq}(TFF) + t_{pd}(CLCG(Our))$ (2). From (1) and (2), we can obtain the speedup as $SPEEDUP = \frac{T_c(Conv)}{T_c(Our)} = \frac{t_{pcq}(DFF)+t_{pd}(CLCG(Conv))+t_{setup}(DFF)}{t_{pcq}(TFF)+t_{pd}(CLCG(Our))}$ (3). If $t_{pcq}(DFF) = t_{pcq}(TFF)$ and $t_{pd}(CLCG(Conv)) = t_{pd}(CLCG(Our))$, the speedup becomes $SPEEDUP = 1 + \frac{t_{setup}(DFF)}{t_{pcq}(TFF)+t_{pd}(CLCG)}$ (4).

If the input of circuit changes during clock at logic one, the possibility exists that this input will generate glitches that can alternate the next stage TFF value. To address this problem, we used pulsed clock signal. The width of the pulsed clock signal is set to be the minimum pulsed clock width of correctly operating TFF. In our experiments the pulse width was 0.1 ns. Since the pulsed clock signal is narrow, the possibility that inputs change during clock at logic one is reduced. In case this very low possibility happens, the width of pulses caused by inputs during clock signal at logic one is always less than the width of the original pulsed clock signal and will not change the state of the TFFs. As a result, the circuit will keep working properly. Another way to handle this clocking issue is to register/synchronize the input with clock signal before it goes to the actual circuit. Since inputs are synchronized, the changing of input during clock at logic one will be ignored by the circuit. However, this requires additional logic area, latency and power overhead. For that reason, we choose to use a narrow size pulsed clock approach in our proposal.

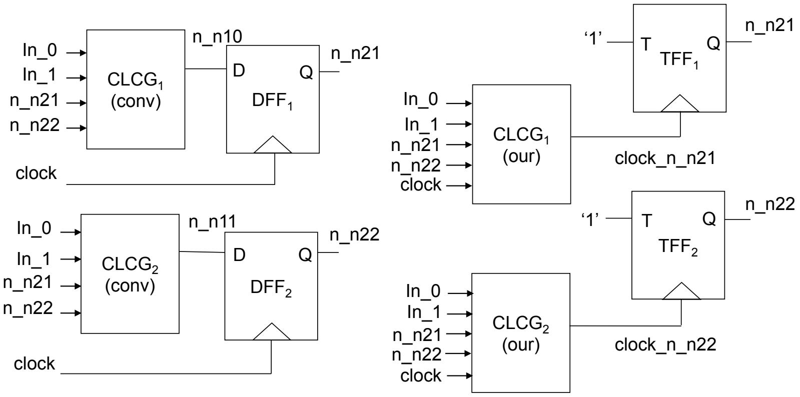
For exemplifying our proposal, we show here an example of how conventional circuits are converted into circuits implemented according to our proposal. Let us assume that we have a conventional circuit as illustrated in Figure 5.4(a) and we want to convert this circuit to our circuit as shown in Figure 5.4(b). In general, we use a simple formula when converting conventional circuits:

$$clock_i(our) = \begin{cases} clock_i(conv) & \text{if } Q_i(conv) \neq D_i(conv) \\ 0 & \text{if } Q_i(conv) = D_i(conv) \end{cases}$$

where:

$clock_i(our)$: clock input of flip-flop i in our circuit;

$clock_i(conv)$: clock input of flip-flop i in conventional circuit;



(a) An example of conventional circuit (b) An example of our circuit

Figure 5.4: Simple circuit examples

$Q_i(\mathtt{conv})$: Q output of flip-flop i in conventional circuit;
$D_i(\mathtt{conv})$: D input of flip-flop i in conventional circuit.

Let us assume that $CLCG_1(\mathtt{conv})$ has the true table as shown in Table 5.1. To convert $CLCG_1(\mathtt{conv})$ to $CLCG_1(\mathtt{our})$, we can capture that $\mathtt{clock}_1(\mathtt{conv}) = \mathtt{clock}$, $D_1(\mathtt{conv}) = \mathtt{n\_n10}$, $Q_1(\mathtt{conv}) = \mathtt{n\_n21}$, and $\mathtt{clock}_1(\mathtt{our}) = \mathtt{clock\_n\_n21}$. By applying the above formula for computing $\mathtt{clock}_i(\mathtt{our})$, we can obtain the true table of $CLCG_1(\mathtt{our})$ for the logic function of $\mathtt{clock\_n\_n21}$ as shown in Table 5.2.

Table 5.1: The true table of $CLCG_1(\mathtt{conv})$

In_0	In_1	n_n21	n_n22	n_n10
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Table 5.2: The true table of $CLCG_1(\mathtt{our})$

In_0	In_1	n_n21	n_n22	clock_n_n21
0	0	0	0	0
0	0	0	1	clock
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	clock
0	1	1	1	clock
1	0	0	0	clock
1	0	0	1	clock
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	clock
1	1	1	1	clock

5.3 Transistor-level Circuit Evaluation

5.3.1 Experimental Setup

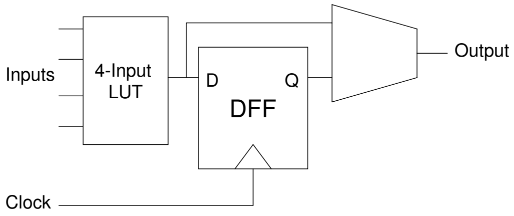
To evaluate the proposed logic element (LE), transistor-level circuit simulations were performed using LTSPICE tools [59] and 45 nm BSIM4 CMOS device models [60] with nominal VDD of 1.2 V. Because we use transistor-level simulation, all internal glitches are implicitly considered. The MCNC benchmark circuits [58] were used for our study. Since our proposal is new, no CAD tools (high level synthesis, technology mapping, place and route tools) are available for targeting FPDs using the proposed LE. For that reason, we performed all design transformations by hand. This is also why we did not evaluate our proposal with all MCNC benchmark circuits; we only evaluated the proposal with the circuits that were not too complex for manual LUT design as shown in Table 5.3. Since our proposal saves power for circuits with storage elements, we selected the representative MCNC benchmark circuits.

Table 5.3: The MCNC benchmark circuits

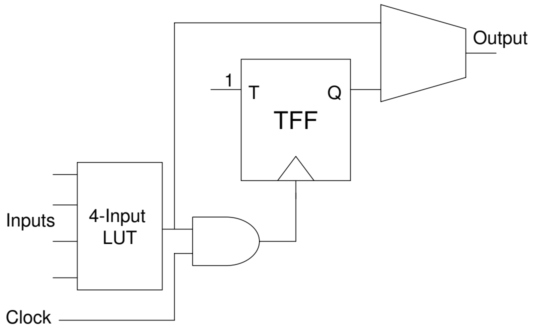
Names	Inputs	Outputs	States	State transitions(STs)	STs to same state
bbtas	2	2	6	24	10
dk27	1	2	7	14	0
lion	2	1	4	11	5
mc	3	5	4	10	5
shiftreg	1	1	8	16	2
tav	4	4	4	49	0
train4	2	1	4	14	7

Due to the fact that SRAM cell values remain constant after configuration (no additional dynamic power) and there is no difference in the number of SRAM cells for FPDs using the conventional and our LEs (same static power), we do not model SRAM in our experiments. We connect the internal signals directly to VDD or ground depending on the intended SRAM content. The simulated nMOS and pMOS transistor dimensions were: Length ($L_n = 45nm$) / Width ($W_n = 90nm$) and $L_p = 45nm$ / $W_n = 270nm$ respectively. The selected ratio between the nMOS and pMOS transistor widths ($\frac{W_p}{W_n} = 3$) is to model the worst case scenario in respect to our proposal when leakage power is considered. To accurately model LUTs, multiplexers, and routing circuits, we selected transmission-gate based implementation as used by Xilinx commercial FPGAs patented in [146]. In this experiment, we assume that unused resources can be turned off to model power gating both for the conventional FPDs and for our proposal.

First, we created experimental circuits representing both for the conventional and the proposed LEs. The experimental LEs are shown in Figure 5.5. In this experiment, an additional AND gate for feeding clock signal was used to make manual implementation of the MCNC circuits easier. An experimental conventional LE consists of a 4-Input LUT, a DFF, and an output multiplexer as illustrated in Figure 5.5(a); while our proposal is represented by a 4-Input LUT, a TFF, an output multiplexer, and an AND gate as shown in Figure 5.5(b). These two LE circuits were used for creating experimental FPD circuits. The flip-flop circuits used for the simulation of the conventional LE and our LE are shown in Figure 5.6 and Figure 5.7. For fair comparison, the only difference between the flip-flop representing our proposal and the conventional flip-flop is the feedback line from the inverted output to the D input. This feedback line forces the flip-flop to behave as a T flip-flop with T input permanently connected to a logic one value.



(a) Conventional logic element



(b) Our logic element

Figure 5.5: Logic elements used in our experiments

Next, the LE circuits were combined with interconnection components to create complete FPD circuits. The interconnection circuits (fixed wires and pro-

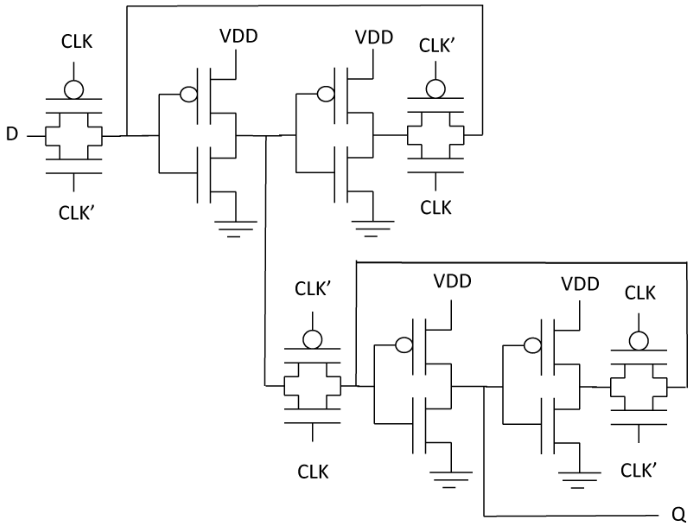


Figure 5.6: A flip-flop circuit used in conventional LE experiments

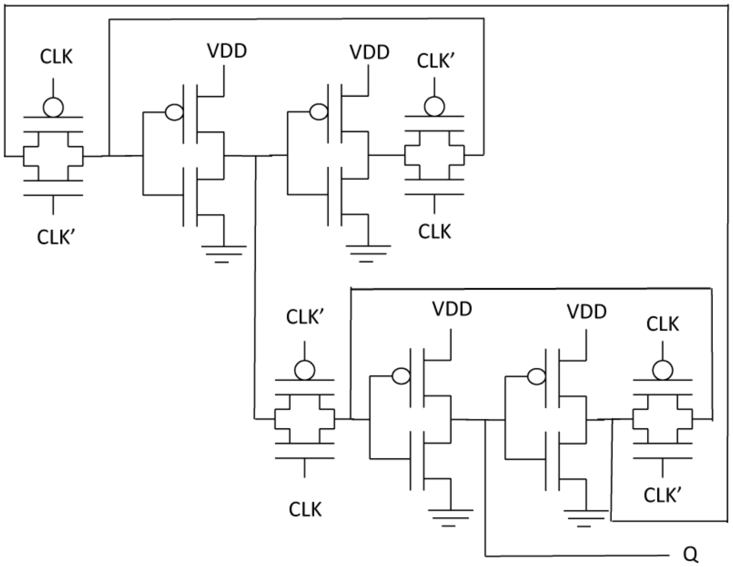


Figure 5.7: A flip-flop circuit used in the proposed LE experiments

grammable switches) were used for connecting needed LE circuits which will be used for creating benchmark circuits.

Finally, we implemented each MCNC benchmark circuit onto the FPDs using conventional LEs and our LEs. MCNC circuits described using Berkeley Logic Interchange Format (BLIF) mapped for 4-input-LUT-based FPDs were used for implementing circuits onto the FPD based on conventional LEs. We manually implemented each MCNC benchmark circuit onto our FPD circuit. In this step, we computed all of the functions needed for the LUTs in the new LEs which are totally different from the functions of the conventional approach. Next, we placed the computed functions in LUTs of the FPD using the proposed LEs and created the required interconnections for each MCNC circuit. The reconfigurations were done by modifying the contents of the 4-Input LUTs, the output multiplexes, and the interconnect control signals.

Before we measured the needed performance parameters, all circuits have been verified to make sure that our circuits perform the same function as the conventional ones by using the same test vectors and the same simulation length. We compared the simulation results of the two implementations. After adopting the pulsed clock in our case, all circuits using our LEs worked properly. The test vectors representing all possible input values combinations were used.

The benchmark circuits were simulated to obtain the needed performance parameters: power, speed, and area for each benchmark circuit. Area is in terms of number of transistors required to implement the benchmark circuit using FPD circuits. The breakdowns of total power which consists of logic power (total power inside LEs), clock power, and interconnect power were also obtained. To make our power study complete, we also analyzed the static and dynamic power. The evaluation was conducted using 500 MHz clock speed representative for the CMOS technology node assumed in our experiments.

5.3.2 Experimental Results

The experimental results in terms of power consumption for FPDs using both conventional and proposed LEs are depicted in Table 5.4 and Table 5.5. The power reduction results presented in Figure 5.8 were computed based on the results from Table 5.4 and Table 5.5. Besides power evaluation, we also investigated the area overhead and the performance improvements of the FPD using the proposed LEs as shown in Figure 5.9.

Table 5.4: Experimental results of Logic, Clock, and Interconnect Power (μW)

Benchmarks	Logic Power		Clock Power		Interconnect Power	
	Conv	Our	Conv	Our	Conv	Our
bbtas	11357	9925	2320	811	2394	1002
dk27	39105	37642	2320	891	4188	2649
lion	5943	4460	1515	540	1507	627
mc	28204	25913	1515	559	2282	1374
shiftreg	3361	2975	2317	804	2171	777
tav	40505	39480	1522	641	3886	2970
train4	5576	4287	1514	538	1475	586

Table 5.5: Experimental results of Dynamic, Static, and Total Power (μW)

Benchmarks	Dynamic Power		Static Power		**Total Power**	
	Conv	Our	Conv	Our	**Conv**	**Our**
bbtas	14461	9650	1610	2088	**16071**	**11738**
dk27	44166	39257	1447	1925	**45613**	**41182**
lion	8047	4390	918	1237	**8965**	**5627**
mc	30444	25970	1557	1876	**32001**	**27846**
shiftreg	6720	2948	1129	1608	**7849**	**4556**
tav	44034	40893	1879	2198	**45913**	**43091**
train4	7647	4174	918	1237	**8565**	**5411**

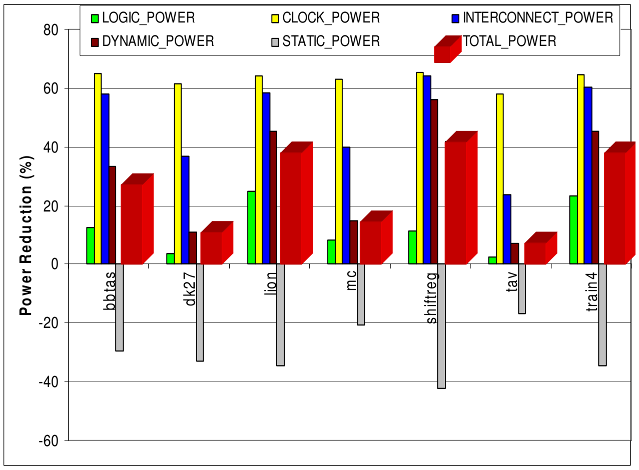


Figure 5.8: Power Reduction (%)

Since the FPD using the proposed LEs avoids unnecessary clock transitions, it consumes up to 65 % less clock power compared to the FPD using conventional LEs as shown in Figure 5.8. By avoiding unnecessary clock transitions, the activity inside the proposed LE is also reduced. As a result, the FPD using proposed LEs has up to 25 % less logic power compared to the FPD using conventional LEs. Our approach also reduces the interconnect activity among LEs resulting in up to 64 % lower interconnect power compared to the FPD using conventional LEs.

The FPD using our proposal reduces up to 56 % dynamic power compared to the FPD using conventional LEs by avoiding unnecessary activities: clock, logic, and interconnect as presented in Figure 5.8. Since the proposed experimental LE has an additional AND gate, the FPD has up to 42 % higher static power as shown in Figure 5.8 and up to 7 % bigger area compared to the FPD using conventional LEs as presented in Figure 5.9. Since not all inputs of LUTs are used in real designs as reported in [21], we can use these unused inputs to feed the clock signal. In this case, we can avoid the additional logic level (the AND gate) for feeding the clock signal.

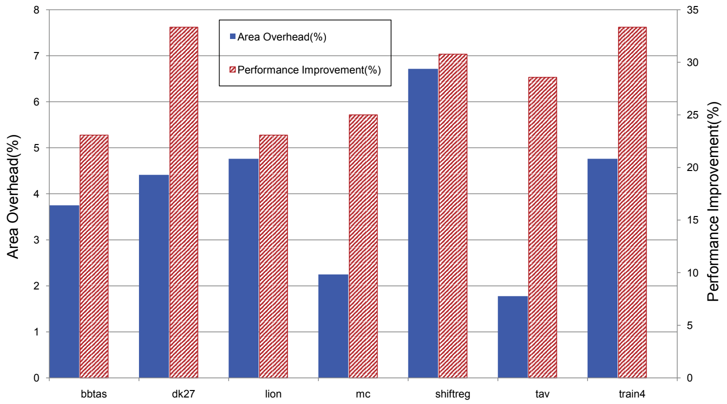


Figure 5.9: Area Overhead and Performance Improvement (%)

Although the FPD using our LEs consumes more static power than the FPD using conventional elements, the overall power consumption of the FPD using our proposal is lower than the conventional one as shown in Figure 5.8. Since the impact of increasing in static power is lower than the impact of reducing the clock, logic, and interconnect powers, the FPD using our proposed LEs still can reduce up to 42 % total power compared to the FPD using conventional LEs as shown in Figure 5.8.

Circuits that do not change their internal state very often will avoid many clock transitions and will be able to achieve more dynamic power reduction compared to circuits that frequently change their states. As shown in Table 5.3, the state of the storage elements in the *dk27* and *tav* benchmark circuits never remains the same. As a result, the total power reduction achieved for these benchmark circuits is smaller compared to other benchmark circuits.

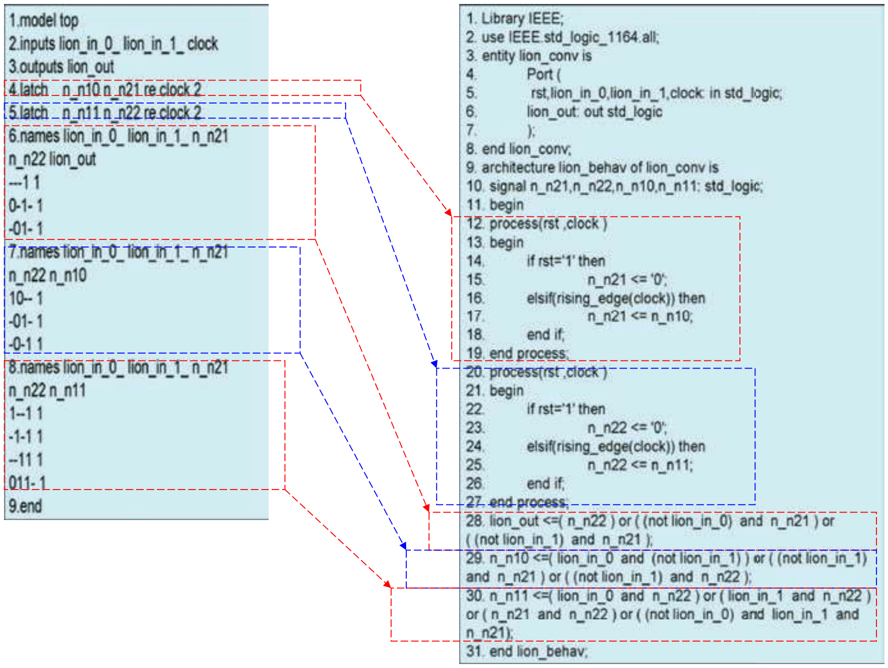
In the conventional LE, the DFF can be clocked by clock signal if only if the D input is ready before the needed setup time for the FF to work properly. In contrast, the TFF in our LE is always ready to receive clock signal since the T input of its TFF is always ready at logic one. As a result, the FPD using proposed LEs runs up to 33 % faster than the FPD using conventional LEs as shown in Figure 5.9.

5.4 Evaluation using a Real CAD Tool on a Real FPGA

5.4.1 Experimental Setup

In this experiment, we force a CAD tool to implement each flip flop with a TFF by proposing a new HDL coding style. To discuss the basic idea of the proposed HDL coding style, an example of MCNC benchmark circuit in Berkeley Logic Interchange Format (BLIF) [147] is presented on the left side of Figure 5.10. This simple example circuit (lion.blif) has two flip-flops (lines 4 and 5) and three combinational logic functions (lines 6-8). Line 6 is the output function; while lines 7 and 8 are the next state functions. In conventional coding style, each flip-flop is coded into one process as shown in the right side of Figure 5.10. This process will generate a D flip-flop (DFF) with the D input from the output of corresponding next state function (NSF) as shown in Figure 5.11(a). In this circuit, the output of each NSF is connected to the D input of DFF. When the D input of DFF (in this figure, for example: n_n10 and n_n11) has a different value compared to its Q output ($D \neq Q$), the DFF needs to be clocked for updating the storage data (in this figure, for example: n_n21 and n_n22). Otherwise, when the D input has the same logic value as the Q output ($D = Q$), the DFF does not need to be clocked. However, since the clock input of DFF is directly connected to the clock signal, the DFF is always clocked. This unnecessary logic transition in this circuit wastes power.

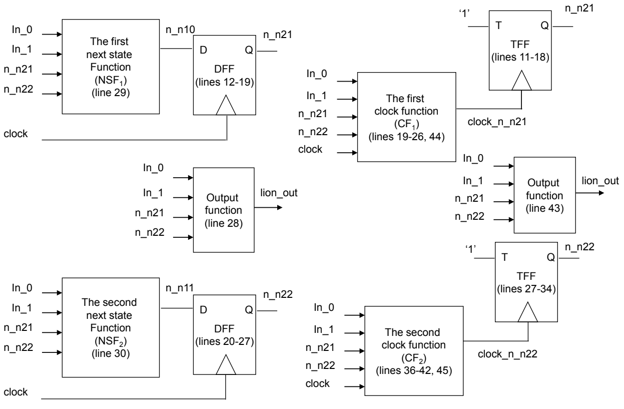
To solve this issue, we propose a new coding style as shown in the right side of Figure 5.12. Contrary to conventional coding style, in our approach, each flip-flop is represented into two processes. The first process is used to implement a T flip-flop (TFF) with T at logic one; while the second process is used to create a function for feeding the clock input of the TFF. We call this function a clock function (CF) as shown in the example of implemented circuit in Figure 5.11(b). The TFF is clocked when it is needed to update storage data (in this



```
1.model top
2.inputs lion_in_0_ lion_in_1_ clock
3.outputs lion_out
4.latch   n_n10 n_n21 re clock 2
5.latch   n_n11 n_n22 re clock 2
6.names lion_in_0_ lion_in_1_ n_n21
n_n22 lion_out
---1 1
0-1- 1
-01- 1
7.names lion_in_0_ lion_in_1_ n_n21
n_n22 n_n10
10-- 1
-01- 1
-0-1 1
8.names lion_in_0_ lion_in_1_ n_n21
n_n22 n_n11
1--1 1
-1-1 1
--11 1
011- 1
9.end
```

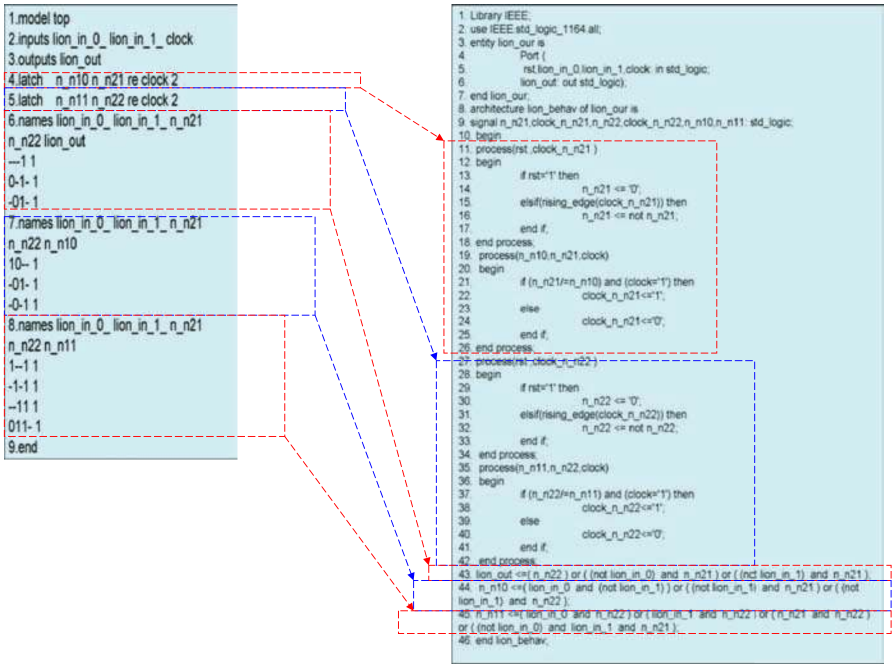
```
1. Library IEEE;
2. use IEEE.std_logic_1164.all;
3. entity lion_conv is
4.          Port (
5.           rst,lion_in_0,lion_in_1,clock: in std_logic;
6.          lion_out: out std_logic
7.          );
8. end lion_conv;
9. architecture lion_behav of lion_conv is
10. signal n_n21,n_n22,n_n10,n_n11: std_logic;
11. begin
12. process(rst ,clock )
13. begin
14.         if rst='1' then
15.                 n_n21 <= '0';
16.         elsif(rising_edge(clock)) then
17.                 n_n21 <= n_n10;
18.         end if;
19. end process;
20. process(rst ,clock )
21. begin
22.         if rst='1' then
23.                 n_n22 <= '0';
24.         elsif(rising_edge(clock)) then
25.                 n_n22 <= n_n11;
26.         end if;
27. end process;
28. lion_out <=( n_n22 ) or ( (not lion_in_0)  and  n_n21 ) or
( (not lion_in_1)  and  n_n21 );
29. n_n10 <=( lion_in_0  and  (not lion_in_1) ) or ( (not lion_in_1)
and  n_n21 ) or ( (not lion_in_1)  and  n_n22 );
30. n_n11 <=( lion_in_0  and  n_n22 ) or ( lion_in_1  and  n_n22 )
or ( n_n21  and  n_n22 ) or ( (not lion_in_0)  and  lion_in_1  and
n_n21);
31. end lion_behav;
```

Figure 5.10: blif to conventional VHDL conversion (lion.blif to lion_conv.vhd)



a) b)

Figure 5.11: lion benchmark example implemented using both coding styles



```
.model top
.inputs lion_in_0_ lion_in_1_ clock
.outputs lion_out
.latch   n_n10 n_n21 re clock 2
.latch   n_n11 n_n22 re clock 2
.names lion_in_0_ lion_in_1_ n_n21
n_n22 lion_out
---1 1
0-1- 1
-01- 1
.names lion_in_0_ lion_in_1_ n_n21
n_n22 n_n10
10-- 1
-01- 1
-0-1 1
.names lion_in_0_ lion_in_1_ n_n21
n_n22 n_n11
1--1 1
-1-1 1
--11 1
011- 1
.end
```

```
Library IEEE;
use IEEE.std_logic_1164.all;
entity lion_our is
            Port (
             rst,lion_in_0,lion_in_1,clock: in std_logic;
            lion_out: out std_logic);
end lion_our;
architecture lion_behav of lion_our is
signal n_n21,clock_n_n21,n_n22,clock_n_n22,n_n10,n_n11: std_logic;
begin
process(rst ,clock_n_n21 )
begin
            if rst='1' then
                        n_n21 <= '0';
            elsif(rising_edge(clock_n_n21)) then
                        n_n21 <= not n_n21;
            end if;
end process;
 process(n_n10,n_n21,clock)
 begin
            if (n_n21/=n_n10) and (clock='1') then
                        clock_n_n21<='1';
            else
                        clock_n_n21<='0';
            end if;
end process;
process(rst ,clock_n_n22 )
begin
            if rst='1' then
                        n_n22 <= '0';
            elsif(rising_edge(clock_n_n22)) then
                        n_n22 <= not n_n22;
            end if;
 end process;
 process(n_n11,n_n22,clock)
 begin
            if (n_n22/=n_n11) and (clock='1') then
                        clock_n_n22<='1';
            else
                        clock_n_n22<='0';
            end if;
 end process;
lion_out <=( n_n22 ) or ( (not lion_in_0)  and  n_n21 ) or ( (not lion_in_1)  and  n_n21 );
 n_n10 <=( lion_in_0  and  (not lion_in_1) ) or ( (not lion_in_1)  and  n_n21 ) or ( (not
lion_in_1)  and  n_n22 );
n_n11 <=( lion_in_0  and  n_n22 ) or ( lion_in_1  and  n_n22 ) or ( n_n21  and  n_n22 )
or ( (not lion_in_0)  and  lion_in_1  and  n_n21 );
end lion_behav;
```

Figure 5.12: blif to our VHDL file conversion (lion.blif to lion_our.vhd)

simple example: n_n21 and n_n22); otherwise, it will not. In this simple circuit, for example if present state of n_n21 is different from next state of n_n21, the TFF will be clocked by clock_n_n21; otherwise, it will not to save power.

The experimental setup is shown in Figure 5.13. Each MCNC benchmark circuit [58] is converted into two VHDL files (conventional and our VHDL files) to represent the two VHDL coding styles (conventional and our coding styles). An example of blif file to conventional VHDL file conversion is presented in Figure 5.10; while an example of blif file to our VHDL file conversion is presented in Figure 5.12.

Each VHDL file is compiled for Stratix EP1S10F484C5 using Compiler Tool from Quartus II. In theVHDL conventional style the D flip-flops are directly connected to the clock signal while in the proposed style the T flip-flops are not. The area needed for implementing each circuit in terms of number of logic elements (LEs) is reported by the Altera Compiler Tool. In the lion benchmark example, both circuits (conventional and our) occupy 7 LEs. The TFF is implemented using the LE with its registered output connected to its input data.

The waveform Editor from Quartus II is used to generate test vectors for each

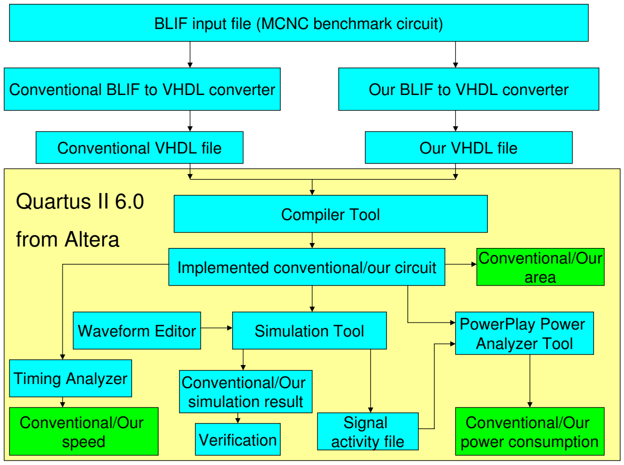

BLIF input file (MCNC benchmark circuit)
Conventional BLIF to VHDL converter
Our BLIF to VHDL converter
Conventional VHDL file
Our VHDL file
Quartus II 6.0 from Altera
Compiler Tool
Implemented conventional/our circuit
Conventional/Our area
Waveform Editor
Simulation Tool
PowerPlay Power Analyzer Tool
Timing Analyzer
Conventional/Our simulation result
Conventional/Our speed
Verification
Signal activity file
Conventional/Our power consumption

Figure 5.13: Experimental setup

benchmark circuit. Those vectors are applied to the implemented circuit using Simulation Tool from Quartus II. Each circuit is verified by comparing the simulation results between the conventional and our circuits. This step is needed to ensure that these two VHDL styles generate functionally correct circuits.

Besides generating simulation results, the Simulation Tool also generates the signal activity file (SAF). To evaluate power consumption, the SAF file and the implemented circuit from the previous step are fed into the Quartus II PowerPlay Power Analyzer Tool to obtain total, dynamic, and static power results.

To compare performance of the implemented circuits, the Timing Analyzer from Quartus II is used. Our study focused on the maximum clock frequency.

5.4.2 Experimental Results

The experimental results of power consumption using a 50 MHz clock are presented in Table 5.6. This table shows that our VHDL style can lead to reduction in dynamic power and total power, but will not reduce static power. Since our VHDL style can avoid unnecessary transitions by clocking flip-flops only when needed, our VHDL style can lead to reduction in dynamic power consumption (75 % on average) compared to conventional VHDL style. The degree of power reduction depends on the nature of the circuit, circuits with many unnecessary transitions can take more advantages of our style in terms of power consumption. This 75 % dynamic power reduction results in only 15 %

average total power consumption reduction at 50 MHz since the static power is dominating. The static power reported by the tools for all of the investigated circuits and both design styles was 187.5 mW.

Table 5.6: Experimental results of power consumption at 50 MHz

Circuits	Dynamic power (mW)			Total power (mW)		
	conv	our	reduction(%)	conv	our	reduction(%)
lion	39.87	4.75	88.09	227.37	192.25	15.45
bbara	36.2	0.78	97.85	223.7	188.28	15.83
bbsse	39.58	6.43	83.75	227.08	193.93	14.6
s298	45.81	10.54	76.99	233.31	198.04	15.12
dk16	51.82	16.93	67.33	239.32	204.43	14.58
dk14	55.58	16.32	70.64	243.08	203.82	16.15
tbk	40.47	4.3	89.37	227.97	191.8	15.87
beecount	44.92	9.56	78.72	232.42	197.06	15.21
cse	41.96	6.48	84.56	229.46	193.98	15.46
s1494	71.73	27.17	62.12	259.23	214.67	17.19
ex1	48.71	16.81	65.49	236.21	204.31	13.5
keyb	41.09	5.21	87.32	228.59	192.71	15.7
planet	42.33	5.88	86.11	229.83	193.38	15.86
pma	89.47	53.51	40.19	276.97	241.01	12.98
s1	52.95	21.23	59.91	240.45	208.73	13.19
styr	66.53	31.55	52.58	254.03	219.05	13.77
s1488	63.96	30.37	52.52	251.46	217.87	13.36
sand	36.14	0.49	98.64	223.64	187.99	15.94

The experiment results of area and performance are presented in Table 5.7. This table shows that our style can also increase the performance of the circuits by 7.6 % on average. This can be explained as following. Since we force CAD tools to implement each flip-flop using a T flip-flop with the T input at logic one in our VHDL style, the flip-flop is always ready to be clocked; it does not need to respect the flip-flop setup time before it can be clocked. Since the setup time is becoming far less significant compared to total longest path for circuits with more logic level, the performance improvement is minimal.

The clock signal needs to be fed to LUTs before it reaches the flip-flops, our style consumes on average 11 % more area compared to the conventional one as shown in this table. If the clock signal can be fed to LUTs using unused inputs, our style does not need additional LUTs for this purpose. As a result, it will produce lower area overhead or even no area overhead as shown in Table 5.7. In our experiment, we had considered this area overhead when we evaluated power consumption and performance.

To investigate all implemented circuits further, we run them using different clock frequencies: 100 MHz, 150 MHz, and 200 MHz. The results of this experiment are presented in Table 5.8. Please note that some of the benchmarks

Table 5.7: Experimental results of area and maximum clock frequency

Circuits	Area (#LEs)			Maximum clock frequency(MHz)		
	conv	our	overhead(%)	conv	our	improvement(%)
lion	7	7	0	437.06	467.07	6.87
bbara	25	29	16	305.44	340.02	11.32
bbsse	45	49	8.89	264.27	274.73	3.96
s298	740	903	22.03	93.82	95.27	1.55
dk16	85	86	1.18	219.97	226.3	2.88
dk14	28	37	32.14	276.78	331.79	19.87
tbk	69	78	13.04	139.14	153.82	10.55
beecount	11	16	45.45	367.92	390.63	6.17
cse	73	80	9.59	216.08	232.34	7.52
s1494	249	261	4.82	190.99	208.9	9.38
ex1	110	118	7.27	242.19	256.41	5.87
keyb	90	96	6.67	190.19	214.5	12.78
planet	215	231	7.44	188.82	210.7	11.59
pma	76	83	9.21	210.39	224.77	6.83
s1	140	146	4.29	117.04	120.44	2.9
styr	202	210	3.96	298.78	310.95	4.07
s1488	243	255	4.94	194.89	197.71	1.45
sand	205	213	3.9	180.08	199.48	10.77

Table 5.8: Power reduction at 100, 150, and 200 MHz

Circuits	Dynamic power reduction (%)			Total power reduction (%)		
	100	150	200	100	150	200
lion	88.09	88.09	88.09	26.28	34.31	40.49
bbara	97.85	97.85	97.85	27.26	35.89	42.64
bbsse	83.75	83.75	83.75	24.86	32.47	38.34
dk16	67.33	67.33	67.33	23.97	30.52	35.35
dk14	70.64	70.64	70.64	26.29	33.25	38.32
tbk	89.37	-	-	26.95	-	-
beecount	78.72	78.72	78.72	25.5	32.92	38.52
cse	84.56	84.56	84.56	26.14	33.97	39.94
s1494	62.12	62.12	-	26.93	33.2	-
ex1	65.49	65.49	65.49	22.39	28.68	33.37
keyb	87.32	87.32	-	26.61	34.64	-
planet	86.11	86.11	-	26.79	34.77	-
pma	40.19	40.19	40.19	19.63	23.66	26.37
s1	59.91	-	-	21.62	-	-
styr	52.58	52.58	52.58	21.82	27.11	30.85
s1488	52.52	52.52	-	21.3	26.56	-
sand	98.64	98.64	-	27.45	36.14	-

did not synthesized at this frequency for both design styles (shown with a dash sign in the table). Since static power, area, and performance are not affected by changing the clock frequency, these tables only show dynamic power and total power consumption results. From these tables, we can observe that dynamic power consumption is linearly proportional to clock frequency. These tables also show that our coding style can reduce total power consumption by 25 %, 32 %, and 36 % on average compared to conventional style at 100 MHz, 150 Mhz and 200 MHz respectively. Since dynamic power is higher when the clock frequency is increased, the reduction of total power is also increased for higher clock frequencies.

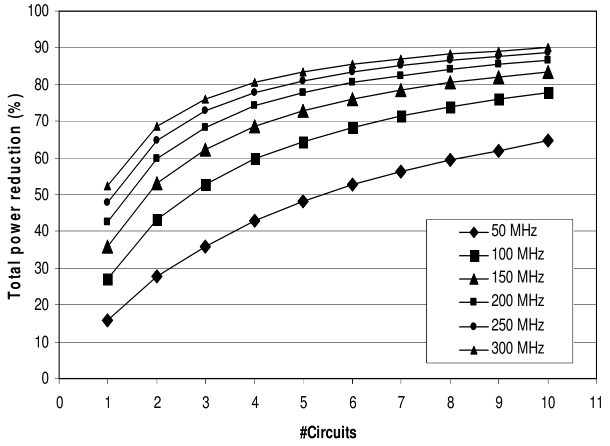


Figure 5.14: Overall power (%) reduction versus number of circuits (#Circuits)

To study the effect of the number of circuits (#Circuits) at different clock frequencies on total power reduction (%), we implement multiple circuits into the FPGA and investigated the effect on overall power reduction as depicted in Figure 5.14. More working circuits means additional dynamic power; the dynamic power becomes more dominant compared to static power. As a result, our coding style reduces more total power when the number of circuits simultaneously implemented on the FPGA increases. This figure indicates that our coding style can reduce total power by 16-65 % at 50 MHz. Total power is significantly reduced at higher frequency, up to 90 % at 300 MHz. Total power reduction saturates as shown in Figure 5.14. This effect is caused by the constant static power contribution that will start dominating the total power number when the number of implemented circuits increases.

5.5 Summary

In this chapter, we have proposed a novel low power logic element (LE) to replace the conventional structures in PLDs and FPGAs. Since unnecessary clock transitions are avoided, the clock power is reduced. By avoiding unnecessary clock transitions, the activity inside the proposed LEs is also reduced. As a result, the FPD using the proposed LEs consumes less logic power compared to the FPD using conventional LEs. Because of activity reduction, the LEs interconnect power is also reduced compared to the FPD using conventional LEs. Moreover, since we do not need an additional controller to hold clock activity, power and area are reduced in comparison to clock gating.

In our LE, since the T input of the FF is always in logic one, the FF is always ready to be clocked. As a consequence, the FPD using our proposed LEs not only consumes less total power by avoiding unnecessary activities: clock, logic, and interconnect, but also runs faster compared to conventional LEs because of its ”always ready” flip-flops.

We also evaluated the proposal using Altera Stratix EP1S10F484C5 and the Quartus II Compiler Tool. To force the tool in implementing circuits according to the proposed LE we used a dedicated coding style. We investigated the gains in power consumption, circuit area and clock frequency. Our approach reduces dynamic power by 75 % at 50MHz but only 15 % in average total power consumption due to the significant contribution of static power.

Table 5.9: Comparison to clock gating solutions

Evaluation	Our coding style solution	Clock Gating solutions		
		[74]	[75]	[76]
Total power reduction	6 - 90 %	5 - 33 %	6.2 - 7.7 %	1.8 - 27.9 %
Performance	2-33 % faster	Not available	0 - 2 % slower	1.1 % faster

Table 5.9 shows the comparison between our solution and clock gating solutions [74] [75] [76]. Clock gating results are obtained from the original papers: [74], [75], and [76]. Unlike clock gating, our proposal does not need an additional controller to stop clock propagation. As a consequence, the FPD using our proposed LEs not only consumes 6 - 90 % less total power by avoiding unnecessary activities: clock, logic, and interconnect, but also it runs 5-33 % faster than traditional clock gating designs. We could not directly compare the area overhead since this information is not reported in the clock gating papers considered. In our case the area overhead varies between 0 and 45 %.

Note. The content of this chapter is based on the the following papers:

T. Marconi, D. Theodoropoulos, K.L.M. Bertels, G. N. Gaydadjiev, **A Novel HDL Coding Style to Reduce Power Consumption for Reconfigurable Devices**, Proceedings of the International Conference on Field-Programmable Technology (FPT), Beijing, China, December 2010.

T. Marconi, K.L.M. Bertels, G. N. Gaydadjiev, **A Novel Logic Element for Power Reduction in FPDs**, CE-TR-2010-01, Computer Engineering Lab, TU Delft, January 2010.

T. Marconi, D. Theodoropoulos, K.L.M. Bertels, G. N. Gaydadjiev, **A Novel HDL Coding Style for Power Reduction in FPGAs**, CE-TR-2010-02, Computer Engineering Lab, TU Delft, January 2010.

6

Improved Configuration Circuit Architecture for FPGAs

L**ong** reconfiguration times form a major bottleneck in dynamic reconfigurable systems. Many approaches have been proposed to address this problem. However, improvements in the configuration circuit that introduces this overhead are usually not considered. The high reconfiguration times are due to the large amount of configuration bits sent through a constrained data path. In order to alleviate this, we propose a novel FPGA configuration circuit architecture to speedup bitstream (re)configuration and relocation. Transporting only the data required for the configuration in flight and avoiding external communication while relocating are two main ideas of our proposal. By utilizing the MCNC benchmark set, the proposal is evaluated against the state of the art approaches in terms of reconfiguration time, relocation time, and bitstream sizes. Moreover, the introduced hardware overhead to support our proposed architecture is also studied.

This chapter is organized as follows. Problem of high reconfiguration overhead in runtime reconfigurable systems is identified in Section 6.1. In Section 6.2, we propose our idea to cope with this high reconfiguration overhead. The proposal is evaluated against related art in Section 6.3. Finally, Section 6.4 ends with the conclusions.

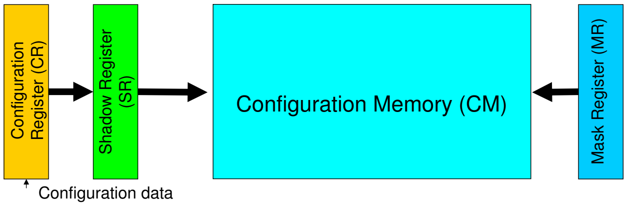
6.1 Introduction

Modern FPGA devices support partial reconfiguration that allows runtime changes of the system functionality. Various benefits can be achieved by exploiting this property, e.g., reduced power consumption, minimized hardware cost, improved system performance, and more [83]. However, there is a problem related to the reconfiguration time penalties that one has to address in order to fully benefit from the above. This problem is even more restrictive for systems where reconfigurations occurs frequently. This high reconfiguration time overheads can eclipse the overall benefits of FPGA based systems. The major bottleneck is introduced by the configuration circuit since it needs to transport large amounts of configuration data (bitstreams) using a limited configuration data path. FPGAs being fine-grained reconfigurable devices, in general require a large amount of configuration bits. In addition, the overall FPGA sizes are also increasing very fast. As a result, the bitstream sizes are growing. Furthermore, to cope with FPGA area fragmentation during runtime, it is needed to efficiently reorganize the positions of the active hardware cores. Such reorganizing process requires fast bitstream relocations.

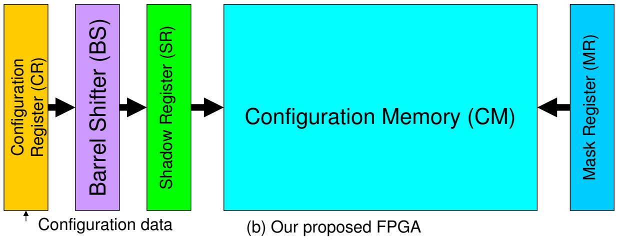
To solve the above problems, in this chapter we propose a novel FPGA configuration circuit architecture to speedup bitstream reconfiguration and relocation. The proposal is evaluated using the Microelectronic Center of North Carolina (MCNC) benchmark circuits [58]. Each benchmark circuit is translated to an equivalent VHDL code before synthesis, mapping, place, and route onto a Xilinx Virtex-4 FPGA. We targeted XC4VLX200-10FF1513 device using Xilinx ISE 8.2.01i_PR_5 tools. Based on the MCNC circuits, our proposal is evaluated against the conventional Virtex-4 reconfiguration process. In terms of relocation times, we compare against a system that modifies the reconfiguration data before sending it to the FPGA configuration circuit during relocation as proposed in [148]- [152]. We also investigate the bitstream size reduction and the hardware overhead of our proposal. The required hardware to implement our architecture is described in VHDL and verified using ModelSim simulation. The verified hardware is synthesized in ASIC with 90 nm CMOS technology using Cadence Encounter tools to obtain the hardware overhead numbers. Considering the fact that no data is available about the exact size of the Xilinx configuration circuit, we compare our proposal against the overall Virtex-4 die size obtained from [153].

6.2 Configuration Circuit Architecture

In [154], Young et al. patented an architecture for FPGA partially reconfiguration referred as the conventional FPGA/architecture in this chapter. The main difference between our proposal (Figure 6.1(b)) and the conventional architecture (Figure 6.1(a)) is the Barrel Shifter (BS). This additional component is the key idea of our architecture that allows us to overcome the limitation of the



(a) Conventional FPGA



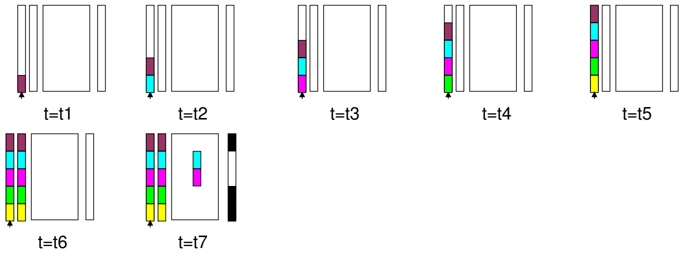
(b) Our proposed FPGA

Figure 6.1: Architecture of the conventional FPGA versus our proposed FPGA

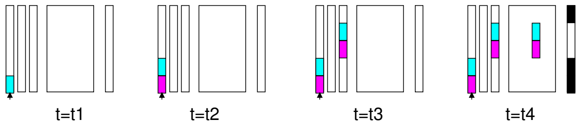
conventional architecture by avoiding shifting and transferring of unnecessary configuration bits.

In the conventional FPGA as presented in [154], a frame of configuration data is loaded serially into a shift register (Configuration Register (CR)) at times t=t1 to t=t5 as illustrated in Figure 6.2(a). After the entire frame is loaded into CR, it is temporarily transferred to a Shadow Register (SR) (Figure 6.2(a) at t=t6) so that the CR is free to begin receiving the next frame of data. An address line is used to transfer the data from the shadow register via the data lines into the selected Configuration Memory (CM) cells as illustrated in Figure 6.2(a) at t=t7. The Mask Register (MR) selects which memory cells receive the specific configuration data values and which do not. This defins a partial reconfiguration zone as shown in the same figure. In this simple example, we can see that although the reconfiguration circuit of the conventional FPGA can partially reconfigure the selected FPGA area, the needed reconfiguration time is still high. This reconfiguration time overhead is due to the shifting of unneeded configuration data (dummy data) along with the needed one.

To overcome the above limitation of conventional FPGAs, we propose to employ a Barrel Shifter (BS) that can prevent of shifting and transferring unnecessary configuration data to CR as illustrated in Figure 6.1(b) and Figure 6.2(b). BS is used to facilitate in transferring the needed configuration data from CR



(a) Conventional FPGA



(b) Our proposed FPGA

Figure 6.2: Reconfiguration steps of the conventional FPGA versus our proposed FPGA

to a specific position in SR. In the proposed FPGA, only the needed configuration data are loaded into CR as shown in Figure 6.2(b) at t=t1 to t2. After a part of the entire frame (the needed configuration data) is shifted into CR, the configuration data are temporarily transferred to SR through BS (Figure 6.2(b) at t=t3). Finally, this configuration data are transferred to selected partial reconfiguration zone defined by MR as illustrated in Figure 6.2(b) at t=t4. Based on this simple example, we can see that our proposed architecture only needs 4 steps instead of 7 steps present in the conventional FPGA in reconfiguring the same partial reconfiguration area.

To support the mechanism of our proposed architecture in shifting configuration data partially from CR register to a specific position in SR register through Barrel Shifter (Figure 6.3), the proposed architecture needs the additional hardware enclosed between the dashed lines in Figure 6.4. The hardware needs to know how much configuration data should be shifted (Frame Height (FH)) and how far the configuration data must be shifted (Frame Height Displacement (FHD)). In order to control FH, we add Frame Height Counter (FHC), Frame Height Register (FHR) and Comparator. This simple structure can control the frame height by writing a specific FH value to FHR during the execution of a set frame height command (FHR initialization). The value of FHC is increased every time the configuration data are shifted into the CR register. When the FHC reaches the same value as FHR, the stop signal from this circuit prevents CR shifting configuration data further. The frame height displacement is controlled by setting the Frame Height Displacement Register (FHDR) using a set height displacement command. This way, we can place the configuration data to the right position into SR in order to partially reconfigure the FPGA with minimal configuration time overhead.

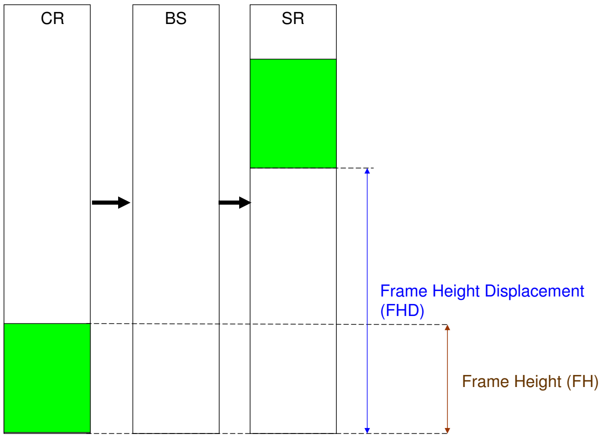


Figure 6.3: Mechanism of our proposed architecture

Instead of including unnecessary dummy data in conventional bitstream for reconfiguration (Figure 6.5(a)), our proposed architecture only includes the required reconfiguration data in the bitstream (Figure 6.5(b)). This feature reduces the configuration bitstream size, consequently it decreases the required memory/hardware for storing the bitstream. In addition to setting the number of frames and the start frame address, before we can write configuration data, we have to set two additional parameters: frame height displacement and frame height using the corresponding set height displacement (to initialize FHDR) and set frame height (to initialize FHR) commands.

Although our proposal can be generally used for any partially reconfigurable FPGAs, we use Virtex-4 [56] as our case study. There are 22 frames per Virtex-4 CLB column and each frame contains 41 words = 41x32 = 1312 bits. Therefore one column CLB needs 22x1312 = 28864 bits. Since one CLB column of Virtex-4 FPGA consists of 16 CLBs, one CLB contains $28864/16 = 1804$

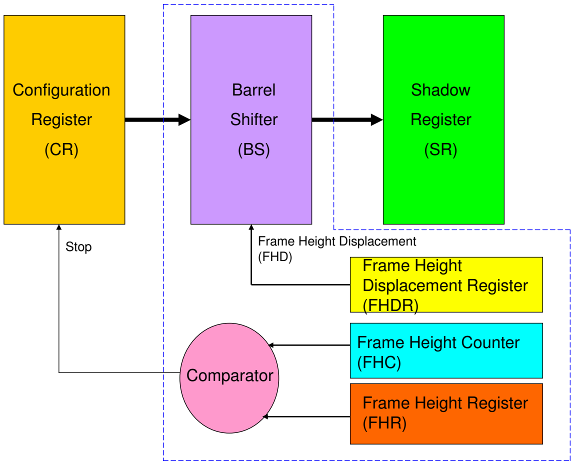
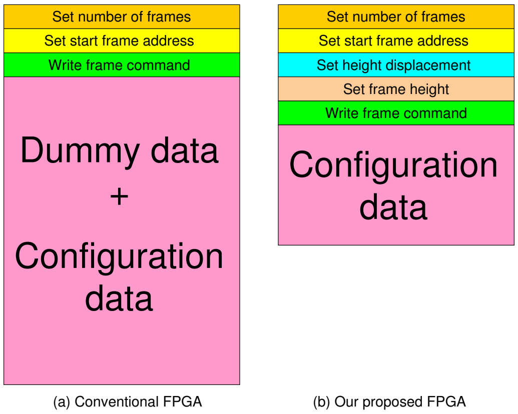

Configuration Register (CR)
Barrel Shifter (BS)
Shadow Register (SR)
Stop
Frame Height Displacement (FHD)
Frame Height Displacement Register (FHDR)
Frame Height Counter (FHC)
Comparator
Frame Height Register (FHR)

Figure 6.4: Hardware overhead for supporting fast reconfiguration

bits. Based on these information and Figure 6.5, the reconfiguration bitstream size in number of bits for the conventional FPGA can be estimated as $RCBSS_{conv} = (3 \times 32) + (28864 \times \lceil h/16 \rceil \times w)$. The factor (3x32) is for set number of frames (32 bits), set start frame address (32 bits), and write frame command (32 bits) as shown in Figure 6.5(a). Since the atomic reconfiguration unit in Virtex-4 is a frame, the $\lceil h/16 \rceil$ is used in this equation. The h and w are the core height and width in number of CLBs. Since the atomic reconfiguration unit in our proposal is a **single** CLB, reconfiguration bitstream size in number of bits for our proposed architecture is estimated as $RCBSS_{our} = (5 \times 32) + (1804 \times h \times w)$. The factor (5x32) is for set number of frames (32 bits), set start frame address (32 bits), set height displacement (32 bits), set frame height (32 bits), and write frame command (32 bits) as shown in Figure 6.5(b). Since Virtex-4 has 32-bit configuration data path, it can transfer 32 bits of data per clock cycle. As a result, the reconfiguration time in number of clock cycles for the conventional architecture is $RCT_{conv} = RCBSS_{conv}/32$. Using the same assumptions, the reconfiguration time in number of clock cycles for our architecture is $RCT_{our} = RCBSS_{our}/32$.

In this chapter, we also propose a new specialized configuration command to support core relocations. Instead of modifying target address and resending the modified reconfiguration data in order to do relocation as proposed by several authors (e.g. [148]- [152]) using the conventional FPGA (Figure 6.6(a)), using



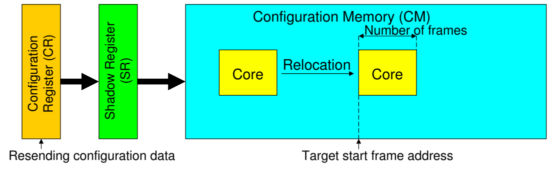
(a) Conventional FPGA (b) Our proposed FPGA

Figure 6.5: Bitstream for reconfiguration

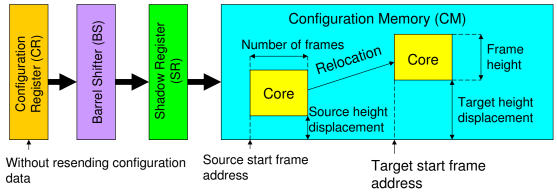
the proposed relocation command provides with the opportunity to perform relocation without resending the reconfiguration data since the reconfiguration data are already in configuration memory (Figure 6.6(b)). To do relocation efficiently, our proposed relocation command copies the relocatable core from configuration memory to the shadow register and then directly writes it back to the target location per frame basis as illustrated in Figure 6.6(b). This new mechanism of relocation makes core relocation faster since we do not need to resend configuration data outside of the FPGA device. This will significantly reduce the configuration path utilization during relocation. Three additional registers are needed to support fast relocation: source start frame address register (SSFAR), target height displacement register (THDR), and source height displacement register (SHDR). To facilitate fast relocation, we need to set additional parameters:

- source start frame address by *set source start frame address* command;
- target height displacement by *set target height displacement* command;
- source height displacement by *set source height displacement* cmd; and
- frame height using a *set frame height* command.

Since the read back capability is already supported by the Virtex-4 devices, no extra hardware is needed to read back configuration memory.


Configuration Register (CR)
Shadow Register (SR)
Configuration Memory (CM)
Number of frames
Core
Relocation
Core
Resending configuration data
Target start frame address

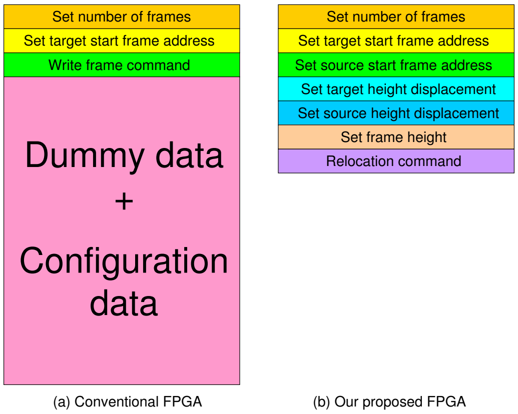
(a) Conventional FPGA


Configuration Register (CR)
Barrel Shifter (BS)
Shadow Register (SR)
Configuration Memory (CM)
Number of frames
Relocation
Core
Core
Frame height
Source height displacement
Target height displacement
Without resending configuration data
Source start frame address
Target start frame address

(b) Our proposed FPGA

Figure 6.6: Core relocation

Based on Virtex-4 FPGA information and Figure 6.7, the relocation bitstream size in number of bits for the conventional FPGA can be estimated as $RLBSS_{conv} = (3 \times 32) + (28864 \times \lceil h/16 \rceil \times w)$ due to the need to resending configuration bitstream. The factor (3x32) is for set number of frames (32 bits), set target start frame address (32 bits), and write frame command (32 bits) as shown in Figure 6.7(a). Since our architecture does not need to resend configuration bitstream during relocation, the relocation bitstream size in number of bits for the our FPGA is constant ($RLBSS_{our} = 7 \times 32$). The factor (7x32) is for set six parameters (number of frames, target start frame address, source start frame address, target height displacement, source height displacement, frame height), and also for the relocation command as shown in Figure 6.7(b). Using 32-bit configuration data path, the relocation time in number of clock cycles for the conventional architecture is $RLT_{conv} = RLBSS_{conv}/32$. Since our proposed relocation command needs to read configuration memory (assume one clock cycle) and write configuration memory (assume another clock cycle) per frame for relocation and each CLB column has 22 frames, the reconfiguration time in number of clock cycles for our architecture is $RLT_{our} = (RLBSS_{our}/32) + (2 \times 22 \times \lceil h/16 \rceil \times w)$.


Set number of frames
Set target start frame address
Write frame command
Dummy data
+
Configuration data
Set number of frames
Set target start frame address
Set source start frame address
Set target height displacement
Set source height displacement
Set frame height
Relocation command

(a) Conventional FPGA (b) Our proposed FPGA

Figure 6.7: Bitstream for core relocation

6.3 Evaluation

To evaluate our proposed architecture, we used the Microelectronic Center of North Carolina (MCNC) benchmark set from [58]. Each benchmark circuit was translated to an equivalent VHDL code before it was synthesized, mapped, placed, and routed onto a Xilinx Virtex-4 FPGA with part number XC4VLX200-10FF1513 using Xilinx ISE 8.2.01i_PR_5 tools. Using these FPGA-implemented MCNC circuits, the proposal was evaluated against the conventional one (Virtex-4 in this case study) in terms of reconfiguration and relocation times and the bitstream sizes.

We assume that the frame utilization (the ratio between number of required bits in a frame and total number of bits in a frame) is equal to the logic utilization. It is well-known that the wire utilization is much lower than the logic utilization. Therefore we consider that the assumption is conservative in that actual frame utilization is significantly lower than the logic utilization. Since square shape circuit performs the best in terms of area and speed as shown in [155], each benchmark circuit was implemented in a free square-shaped-area of the FPGA.

Table 6.1 shows the speedup (times) and bitstream size (BSS) reductions (in %) of the proposed architecture compared to the conventional Xilinx architecture in terms of both, reconfiguration and relocation times. From this table, we can see that the proposed architecture has on average 4 times shorter reconfiguration times. The reconfiguration time reduction (state as speedup S

Table 6.1: Speedup (S) (times) and bitstream size (BSS) reduction (R) (%) compared to the conventional FPGA (h and w are in number of occupied CLBs)

Circuit	h	w	RCT			BSS			RLT		
			Conv	Our	S	Conv	Our	R(%)	Conv	Our	S
s27	1	1	905	62	14.6	28960	1964	93.22	905	51	17.75
mult16b s208.1	2	2	1807	231	7.82	57824	7376	87.24	1807	95	19.02
s344, s349 s526, s526n s382, s386 s400, s420.1 s444, mm4a s641	3	3	2709	513	5.28	86688	16396	81.09	2709	139	19.49
mult16a, s713 s510, s828.1	4	4	3611	907	3.98	115552	29024	74.88	3611	183	19.73
s820, mm9a s832	5	5	4513	1415	3.19	144416	45260	68.66	4513	227	19.88
s1196, mm9b mult32a s1488, s1494	6	6	5415	2035	2.66	173280	65104	62.43	5415	271	19.98
sbc, s1423 mm30a s9234.1	7	7	6317	2768	2.28	202144	88556	56.19	6317	315	20.05
s5378	9	9	8121	4572	1.78	259872	146284	43.71	8121	403	20.15
s298	10	10	9023	5643	1.6	288736	180560	37.47	9023	447	20.19
dsip	11	11	9925	6827	1.45	317600	218444	31.22	9925	491	20.21
bigkey	14	14	12631	11055	1.14	404192	353744	12.48	12631	623	20.27
clma	20	20	36083	22555	1.6	1154656	721760	37.49	36083	1767	20.42
s38584.1	24	24	43299	32477	1.33	1385568	1039264	24.99	43299	2119	20.43
s38417	25	25	45103	35240	1.28	1443296	1127660	21.87	45103	2207	20.44

in the table) can improve up to 14.6 times for small circuits (s27) since the amount of dummy data increases when the circuit size is smaller. In this case, the conventional architecture is very slow in reconfiguring the FPGA fabric, while our proposal becomes very efficient since we do not need to transport unneeded data. Moreover, the reconfiguration bitstream size is reduced by up to 93.22 % compared to the conventional FPGA since we do not include any dummy data in our bitstream file. Please note that s820 and s832 results are almost similar although they represent two completely different designs. This is due to their similar sizes when mapped to Xilinx FPGA and measured in terms of used CLBs, which is the atomic measurement unit for our study.

Table 6.1 also shows that our proposal relocates cores faster than the conventional architecture. On average, our proposal can do relocation 19.8 times faster than the conventional FPGA by avoiding resending configuration data during relocation using the proposed specialized relocation command.

Besides, the hardware overhead of our proposal was also evaluated as depicted in Table 6.2. The required hardwares to build our proposed architecture was coded in VHDL and verified using ModelSim simulation. Since the compared FPGA (Virtex-4) was implemented using 90 nm technology, to be fair in comparison, the verified hardware was also implemented in ASIC with 90 nm CMOS technology using Cadence Encounter tools to obtain our hard-

Table 6.2: Area overhead

Modules	Area(μm^2)
Barrel Shifter (BS)	61881
Comparator	450
Frame Height Displacement Register(FHDR)	1092
Frame Height Counter (FHC)	1650
Frame Height Register (FHR)	1092
Source Start Frame Address Register (SSFAR)	1092
Target Height Displacement Register (THDR)	1092
Source Height Displacement Register (SHDR)	1092
Total Area Overhead	69441

ware overhead. To compare our proposal against Virtex-4 in terms of area, the estimated die size of Virtex-4 FPGA was obtained from [153]. Considering the area(=735mm^2) of the targeted Virtex-4 FPGA device (Estimated die size from [153]), the area overhead of the architecture is very small. The total area overhead is only 0.009 % of the Virtex-4 area.

6.4 Summary

In this chapter, we have introduced a novel configuration circuit architecture for partially reconfigurable FPGAs, that supports faster bitstream reconfiguration and relocation. More precisely, our proposal is 4x faster during reconfiguration of the MCNC benchmark circuits compared to Xilinx Virtex-4. In addition, the area overhead of the proposed architecture is only 0.009 % of the overall Virtex-4 area. For fast 2D relocation, we proposed a new specialized command in the configuration protocol. With this new command, hardware core relocation is facilitated without resending configuration data externally. Our experimental results show that our architecture is 19.8x faster during relocation compared to the current state the art (Virtex-4). Moreover, the bitstream sizes of the investigated MCNC benchmarks are reduced by 65 % on average when our approach is applied.

Note. The content of this chapter is based on the the following paper:

T. Marconi, J.Y. Hur, K.L.M. Bertels, G. N. Gaydadjiev, **A Novel Configuration Circuit Architecture to Speedup Reconfiguration and Relocation for Partially Reconfigurable Devices**, Proceedings of IEEE Symposium on Application Specific Processors (SASP), June 2010.

7

Conclusions and Future Work

In this dissertation, novel proposals for dealing with the main problems in runtime reconfigurable systems have been presented. More specific, efficient online hardware task scheduling and placement, power consumption reduction and runtime reconfiguration overhead reduction have been addressed. The proposals have been evaluated against existing state of the art solutions. The main contributions of the thesis are summarized in Section 7.1 and future directions are presented in Section 7.2.

7.1 Main Contributions

In the context of runtime reconfigurable systems on partially reconfigurable devices, the main contributions of this dissertation can be summarized as follows.

1. Two novel algorithms, called Intelligent Merging(IM) and Quad-Corner(QC), for online placement of reconfigurable hardware tasks on partially reconfigurable devices have been presented. Because of its on-demand merging capability, the IM can speedup online placement algorithms by 1.72x while loosing only 0.89 % placement quality on average. Experiments with real hardware tasks on Virtex-4 show that the QC not only has 78 % less penalty and 93 % lower wasted area than the existing algorithms on average due to its quad-corner spreading capability but also has 86 % lower runtime overhead due to its simplicity.
2. Two novel online hardware task scheduling and placement algorithms have been proposed. The first algorithm, Intelligent Stuffing(IS), is designed for 1D area model, whereas the second one, 3D Compaction (3DC), is proposed for 2D area model. Because of having the additional

alignment status, the IS outperforms the existing algorithms in terms of reduced total wasted area up to 89.7%, has 1.5 % shorter schedule time and 31.3% faster response time. Due to the blocking-awareness, the 3DC not only has better scheduling and placement quality (up to 4.8 % shorter schedule time, 75.1 % lower waiting time, and 22.9 % less wasted volume) but also has 97 % lower runtime overhead compared to existing algorithms reported in the literature.

3. A novel low power logic element (LE) for FPDs to replace the conventional LE has been proposed and carefully evaluated. The FPDs using our proposal have 6-90 % lower total power due to the avoidance of unnecessary activities. Devices using our LE run 2-33 % faster than conventional systems due to our ”always ready” LE flip-flops.

4. A novel configuration circuit architecture for partially reconfigurable FPGAs has been proposed. The proposal reconfigures FPGAs 4x faster by avoiding sending unnecessary data and relocates hardware cores 19.8x faster due to its specialized command compared to Xilinx Virtex-4 with only 0.009 % area overhead. Moreover, the bitstream sizes of the investigated MCNC benchmarks are reduced by 65 % on average when our approach is applied.

7.2 Open Issues

The following open issues can be considered for future work on the topic.

1. Dynamic power is linearly proportional to the clock frequency. Running hardware tasks at lower clock speeds can reduce power consumption. In many cases, some tasks can be operated at lower speeds without affecting on the overall system performance. Hence, online task scheduling and placement algorithms that can run hardware tasks at different clock speeds to reduce power consumption without sacrificing performance can be considered as a future work.

2. A simple way to reduce both static and dynamic power consumptions is to scale down the supply voltage. Online task scheduling and placement algorithms that can run hardware task at different supply voltages should be investigated for power reduction.

3. FPGAs with built-in online scheduling and placement hardware can be studied for future general purpose computing systems.

4. When the chip technology is scaled down, it is becoming almost impossible to create chips without any defects. Defect-aware online task scheduling and placement algorithms that can intelligently manage the defected chips are worthy to be investigated.

5. To reduce reconfiguration overhead further, utilizing placed hardware tasks by reusing them for serial executions or copying them for parallel executions can be integrated in our framework for future study. This technique is supported by fast relocation of our reconfiguration infrastructure in Chapter 6.

6. Designing circuits targeting FPDs based on our proposed low power LEs was performed by hand in this dissertation. To make this design process automatically, CAD tools development for FPDs targeting our proposed LEs is needed to be investigated further.

7. Benefits of replacing FFs with latches are increased performance, area reduction, and minimized power consumption as have been investigated in ASIC designs. An interesting research direction is to study of replacing FFs with latches in FPDs targeting our proposed LEs.

8. Hardware tasks can be placed more efficient if we can have more flexibility in rotating the tasks during reconfiguration. FPGAs with reconfiguration circuit that supports this flexibility is a challenge for future research. This requires homogeneity of the reconfigurable fabric.

9. If the homogeneity presents in FPGAs as required in previous future direction, it is interesting to investigate in what extend this homogeneity will affect the overall system performance. There will be a trade-off between flexibility and performance. ”Can this flexibility cope with its performance degradation?” is another future question that need to be studied further.

Bibliography

[1] P. Lysaght and J. Dunlop, ”Dynamic Reconfiguration of FPGAs,” Proceedings of the International Workshop on Field Programmable Logic and Applications, pp. 82-94, Sept. 1993.

[2] H. Eggers, P. Lysaght, H. Dick, and G. McGregor, ”Fast Reconfigurable Crossbar Switching in FPGAs,” Proceedings of the International Workshop on Field-Programmable Logic, Smart Applications, New Paradigms and Compilers, pp. 297-306, 1996.

[3] R. D. Hudson, D. I. Lehn, and P. M. Athanas, ”A Run-Time Reconfigurable Engine for Image Interpolation,” Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines, pp. 88-95, 1998.

[4] J. Villasenor, C. Jones, and B. Schoner, ”Video Communications Using Rapidly Reconfigurable Hardware,” IEEE Transactions on Circuits and Systems for Video Technology, Vol. 5, No. 6, pp. 565-567, Dec. 1995.

[5] C. Patterson, ”A Dynamic FPGA Implementation of the Serpent Block Cipher,” Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems, pp. 141-155, 2000.

[6] J.G. Eldredge and B.L. Hutchings, ”Density Enhancement of a Neural Network Using FPGAs and Run-Time Reconfiguration,” Proceeding of IEEE workshop on FPGAs for custom computing machines, pp. 180-188, 1994.

[7] M.J. Wirthlin and B.L. Hutchings, ”Sequencing Run-Time Reconfigured Hardware with Software,” Proceedings of ACM/SIGDA International Symposium on Field Programmable Gate Arrays, pp. 122-128, 1996.

[8] S. Govindarajan, I. Ouaiss, M. Kaul, V. Srinivasan, and R. Vemuri, ”An Effective Design System for Dynamically Reconfigurable Architectures,” Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines, pp. 312-313, 1998.

[9] A. Derbyshire and W. Luk, ”Combining Serialisation and Reconfiguration for FPGA Designs,” Proceedings of Field-Programmable Logic and Applications (FPL), pp. 636-645, 2000.

[10] S.R. Park and W. Burleson, ”Reconfiguration for Power Saving in Real-time Motion Estimation,” Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing, pp. 3037-3040, 1998.

[11] S. Toscher, R. Kasper, and T. Reinemann, ”Dynamic Reconfiguration of Mechatronic Real-Time Systems Based on Configuration State Machines,” Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, pp. 172-175, 2005.

[12] R. Tessier, S. Swaminathan, R. Ramaswamy, D. Goeckel, and W. Burleson, ”A Reconfigurable, Power-Efficient Adaptive Viterbi Decoder,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 13, No. 4, pp. 484-488, 2005.

[13] J. Castillo, P. Huerta, C. Pedraza, and J. I. Martínez, ”A Self-Reconfigurable Multimedia Player on FPGA,” Proceedings of IEEE International Conference on Reconfigurable Computing and FPGA’s, pp. 1-6, 2006.

[14] J. Noguera and I. O. Kennedy, ”Power Reduction in Network Equipment through Adaptive Partial Reconfiguration,” Proceedings of Field-Programmable Logic and Applications (FPL), pp. 240-245, 2007.

[15] R. Hymel, A.D. George, and H. Lam, ”Evaluating Partial Reconfiguration for Embedded FPGA Applications,” Proceedings of High-Performance Embedded Computing (HPEC) Workshop, pp. 1-3, Sept. 2007.

[16] A. Haase, C. Kretzschmar, R. Siegmund, D. Müller, J. Schneider, M. Boden, and M. Langer, ”Design of Reed Solomon Decoder using Partial Dynamic Reconfiguration of Xilinx Virtex FPGAs - a Case Study,” Proceedings of Design, Automation and Test in Europe, pp. 151-157, Mar. 2002.

[17] S. Brown and J. Rose, ”Architecture of FPGAs and CPLDs: A Tutorial,” IEEE Design and Test of Computers, Vol. 13, No. 2, pp. 42-57, 1996.

[18] I. Kuon and J. Rose, ”Measuring the Gap between FPGAs and ASICs,” Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays, Monterey, California, USA, pp. 21-30, 2006.

[19] E. I. Boemo , G. G. de Rivera , S. López-Buedo , and J. M. Meneses, ”Some Notes on Power Management on FPGA-Based Systems,” Proceedings of the 5th International Workshop on Field-Programmable Logic and Applications (FPL), pp. 149-157, 1995.

[20] A. Gayasen, K. Lee, V. Narayanan, M. Kandemir, M. J. Irwin, and T. Tuan, ”A Dual-Vdd Low Power FPGA Architecture,” Proceedings of the International Conference on Field-Programmable Logic and its Applications (FPL), pp. 145-157, August 2004.

[21] S. Mondal and S. O. Memik, ”Fine-grain Leakage Optimization in SRAM based FPGAs,” Proceedings of the ACM Great Lakes Symposium on VLSI, pp. 238-243, 2005.

[22] K. Bazargan, R. Kastner, and M. Sarrafzadeh, ”Fast Template Placement for Reconfigurable Computing Systems,” IEEE Design and Test of Computers, vol. 17, pp. 68-83, 2000.

[23] J. Tabero, J. Septién, H. Mecha, and D. Mozos, ”A Low Fragmentation Heuristic for Task Placement in 2D RTR HW Management,” Proceedings of Field-Programmable Logic and Applications (FPL), pp. 241-250, 2004.

[24] J. Tabero, H. Wick, J. Septién, and S. Roman, ”A Vertex-List Approach to 2D HW Multitasking Management in RTR FPGAs,” Proceedings of Conference on Design of Circuits and Integrated Systems (DCIS), pp. 545-550, November 2003.

[25] J. Tabero, J. Septién, H. Mecha, and D. Mozos, ”Task Placement Heuristic Based on 3D-Adjacency and Look-Ahead in Reconfigurable Systems,” Proceedings of Asia and South Pacific Design Automation Conference, pp. 396-401, 2006.

[26] C. Steiger, H. Walder, M. Platzner, and L. Thiele, ”Online Scheduling and Placement of Real-time Tasks to Partially Reconfigurable Devices,” Proceedings of Real-Time Systems Symposium (RTSS), pp. 224-225, Dec 2003.

[27] H. Walder, C. Steiger, and M. Platzner, ”Fast Online Task Placement on FPGAs: Free Space Partitioning and 2D-hashing,” Proceedings of International Parallel and Distributed Processing Symp. (IPDPS), p.178b, April 2003.

[28] M. Morandi, M. Novati, M.D. Santambrogio, and D. Sciuto, ”Core Allocation and Relocation Management for a Self Dynamically Reconfigurable Architecture,” Proceedings of IEEE Computer Society Annual Symposium on VLSI, pp. 286-291, 2008.

[29] A. Ahmadinia, C. Bobda, and J. Teich, ”A Dynamic Scheduling and Placement Algorithm for Reconfigurable Hardware,” Proceedings of Architecture of Computing Systems (ARCS), pp. 125-139, 2004.

[30] M. Handa and R. Vemuri, ”An Efficient Algorithm for Finding Empty Space for Online FPGA Placement,” Proceedings of Design Automation Conference (DAC), pp. 960-965, June 2004.

[31] M. Handa and R. Vemuri, ”An Integrated Online Scheduling and Placement Methodology,” Proceedings of International Conference on Field Programmable Logic and Applications (FPL), pp. 444-453, Aug./Sept. 2004.

[32] J. Cui, Q. Deng, X. He, and Z. Gu, ”An Efficient Algorithm for Online Management of 2D Area of Partially Reconfigurable FPGA,” Proceedings of Design Automation and Test in Europe (DATE), pp. 129-134, Apr. 2007.

[33] Y. Xiao, Z. Duan, and P. Nie, ”An Efficient Algorithm for Finding Empty Space for Reconfigurable Systems,” Proceedings of the 2009 Third IEEE International Symposium on Theoretical Aspects of Software Engineering, pp. 36-43, 2009.

[34] J. Cui, Z. Gu, W. Liu, and Q. Deng, ”An Efficient Algorithm for Online Soft Real-Time Task Placement on Reconfigurable Hardware Devices,” Proceedings of IEEE International Symposium on Object/component/services-oriented Real-time Distributed Computing (ISORC), pp. 321-328, May 2007.

[35] M. Tomono, M. Nakanishi, S. Yamashita, K. Nakajima, and K. Watanabe, ”A New Approach to Online FPGA Placement,” Proceedings of Conference of Information Science and Systems (CISS), pp. 145-150, March 2006.

[36] M. Tomono, M. Nakanishi, S. Yamashita, K. Nakajima, and K. Watanabe, ”An Efficient and Effective Algorithm for Online Task Placement with I/O Communications in Partially Reconfigurable FPGAs,” IEICE

Trans. Fundamentals, Vol. E89-A, No. 12, pp. 3416-3426, December 2006.

[37] A. Ahmadinia, M. Bednara, C. Bobda and J. Teich, ”A New Approach for On-line Placement on Reconfigurable Devices,” Proceedings of International Parallel and Distributed Processing Symposium (IPDPS), pp. 134-140, April 2004.

[38] A. Ahmadinia, C. Bobda, D. Koch, M. Majer and J. Teich, ”Task Scheduling for Heterogeneous Reconfigurable Computers,” Proceedings of Symposium on Integrated Circuits and Systems Design (SBCCI), pp. 22-27, September 2004.

[39] C-H. Lu, H-W. Liao, and P-A. Hsiung, ”Multi-objective Placement of Reconfigurable Hardware Tasks in Real-Time System,” Proceedings of International Conference on Computational Science and Engineering, vol. 2, pp.921-925, 2009.

[40] M. Elbidweihy and J. L. Trahan, ”Maximal Strips Data Structure to Represent Free Space on Partially Reconfigurable FPGAs,” International Journal of Parallel, Emergent and Distributed Systems, Vol. 24, No. 4, pp. 349-366, August 2009.

[41] A. Ahmadinia, C. Bobda, S. Fekete, J. Teich and J. van der Veen, ”Optimal Routing-Conscious Dynamic Placement for Reconfigurable Devices,” Proceedings of International Conference on Field-Programmable Logic and Applications (FPL), LNCS 3203, pp. 847-851, 2004.

[42] A. Ahmadinia, C. Bobda, S. Fekete, J. Teich and J. van der Veen, ”Optimal Free-space Management and Routing-conscious Dynamic Placement for Reconfigurable Devices,” IEEE Transactions on Computers, Vol. 56, No. 3, pp. 673-680, 2007.

[43] M. Köster, M. Porrmann, and H. Kalte, ”Task Placement for Heterogeneous Reconfigurable Architectures,” Proceedings of IEEE International Conference on Field-Programmable Technology (FPT), pp. 43-50, December 2005.

[44] A. Ahmadinia and J. Teich, ”Speeding up Online Placement for XILINX FPGAs by Reducing Configuration Overhead,” Proceedings of IFIP International Conference on VLSI-SOC, pp. 118-122, December 2003.

[45] A.A. ElFarag, H.M. El-Boghdadi, and S.I. Shaheen, ”Miss Ratio Improvement For Real-Time Applications Using Fragmentation-Aware Placement,” Proceedings of Parallel and Distributed Processing Symposium (IPDPS), pp. 1-8, March 2007.

[46] C. Steiger, H. Walder, and M. Platzner, ”Heuristics for Online Scheduling Real-Time Tasks to Partially Reconfigurable Devices,” Proceeding of Field-Programmable Logic and Applications (FPL), LNCS 2778, pp. 575-584, 2003.

[47] C. Steiger, H. Walder, and M. Platzner, ”Operating Systems for Reconfigurable Embedded Platforms: Online Scheduling of Real-Time Tasks,” IEEE transaction on Computers, Vol. 53, No. 11, pp. 1393-1407, 2004.

[48] Y. Chen and P. Hsiung, ”Hardware Task Scheduling and Placement in Operating Systems for Dynamically Reconfigurable SoC,” Proceeding of IFIP International Conference on Embedded and Ubiquitous Computing (EUC), LNCS 3824, pp. 489-498, 2005.

[49] T. Marconi, Y. Lu, K.L.M. Bertels, and G. N. Gaydadjiev, ”Online Hardware Task Scheduling and Placement Algorithm on Partially Reconfigurable Devices,” Proceedings of International Workshop on Applied Reconfigurable Computing (ARC), pp. 306-311, London, UK, March 2008.

[50] Y. Lu, T. Marconi, K.L.M. Bertels, and G. N. Gaydadjiev, ”Online Task Scheduling for the FPGA-Based Partially Reconfigurable Systems,” Proceedings of International Workshop on Applied Reconfigurable Computing (ARC), pp. 216-230, March 2009.

[51] X. Zhou, Y. Wang, X. Huang, and C. Peng, ”On-line Scheduling of Real-time Tasks for Reconfigurable Computing System,” Proceedings of the International Conference on Field-Programmable Technology (FPT), pp. 57-64, 2006.

[52] X. Zhou,Y. Wang, X. Huang, and C. Peng, ”Fast On-line Task Placement and Scheduling on Reconfigurable Devices,” Proceeding of Field-Programmable Logic and Applications (FPL), pp. 132-138, 2007.

[53] D.D. Sharma and D.K. Pradhan, ”A Fast and Efficient Strategy for Submesh Allocation in Mesh-Connected Parallel Computers,” Proceedings

of the 22nd Annual International Parallel and Distributed Processing Symposium (IPDPS), pp. 682-689, 1993.

[54] R. J. Meeuws, Y. D. Yankova, K.L.M. Bertels, G. N. Gaydadjiev, and S. Vassiliadis, ”A Quantitative Prediction Model for Hardware/Software Partitioning,” Proceedings of Field-Programmable Logic and Applications (FPL), pp. 735-739, August 2007.

[55] Y. D. Yankova, K. Bertels, S. Vassiliadis, R. J. Meeuws, and A. Virginia, ”Automated HDL Generation: Comparative Evaluation,” Proceedings of the IEEE International Symposium on Circuits and Systems, pp. 2750-2753, May 2007.

[56] Xilinx, ”Virtex-4 FPGA Configuration User Guide,” UG071, 2008.

[57] J. M. Birkner and H-T. Chua, ”Programmable Array Logic Circuit,” US Patent No. 4124899, 1978.

[58] S. Yang, ”Logic Synthesis and Optimization Benchmarks User Guide Version 3.0,” Technical Report, Microelectronic Center of North Carolina (MCNC), 1991.

[59] http://www.linear.com/designtools/software/

[60] http://www.eas.asu.edu/ ptm/latest.html

[61] M. Klein, ”Power Consumption at 40 and 45 nm,” Xilinx WP298, April 2009.

[62] P. Abusaidi, M. Klein, and B. Philofsky, ”Virtex-5 FPGA System Power Design Considerations,” Xilinx WP285 (v1.0) February 14, 2008.

[63] Altera, ”FPGA Architecture,” WP-01003, July 2006.

[64] S.J.E. Wilton, S.S. Ang, and W. Luk, ”The impact of pipelining on energy per operation in field programmable gate arrays,” Proceedings of Field Programmable Logic and Applications, LNCS 3203, pp. 719-728, 2004.

[65] S. Bard and N. I. Rafla, ”Reducing Power Consumption in FPGAs by Pipelining,” Proceedings of Circuits and Systems, pp. 173-176, 2008.

[66] N. Rollins and M. J. Wirthlin, ”Reducing Energy in FPGA Multipliers through Glitch Reduction,” Proceedings of International Conference

on Military Applications of Programmable Logic Devices, pp. 1-10, September 2005.

[67] R. Fischer, K. Buchenrieder, and U. Nageldinger, ”Reducing the Power Consumption of FPGAs through Retiming,” Proceedings of the International Conference and Workshops on Engineering of Computer-Based Systems, pp.89-94, April 2005.

[68] J. Lamoureux, G.G. Lemieux, and S.J.E. Wilton, ”GlitchLess: Dynamic Power Minimization in FPGAs through Edge Alignment and Glitch Filtering,” Transactions on Very Large Scale Integration Systems (TVLSI), Vol. 16, No. 11, pp. 1521-1534, 2008.

[69] Q. Dinh, D. Chen, and M.D.F. Wong, ”A Routing Approach to Reduce Glitches in Low Power FPGAs,” Proceedings of IEEE/ACM International Symposium on Physical Design, pp. 99-106, March 2009.

[70] L. Cheng, D. Chen, and M.D.F. Wong, ”GlitchMap: An FPGA Technology Mapper for Low Power Considering Glitches,” Proceedings of Design Automation Conference, pp. 318-323, June 2007.

[71] T. Czajkowski and S. D. Brown, ”Using Negative Edge Triggered FFs to Reduce Glitching Power in FPGA Circuits,” Proceedings of Design Automation Conference, pp. 324-329, June 2007.

[72] A. A. Gaffar, J. A. Clarke, and G. A. Constantinides, ”PowerBit - Power Aware Arithmetic Bit-width Optimization,” Proceedings of IEEE International Conference on Field Programmable Technology, pp. 289-292, December 2006.

[73] G. Constantinides, ”Word-length Optimization for Differentiable Nonlinear Systems,” ACM Trans. on Design Automation of Electronic Sys., Vol. 11, No. 1, pp. 26-43, 2006.

[74] Y. Zhang, J. Roivainen, and A. Mämmelä, ”Clock-Gating in FPGAs: A Novel and Comparative Evaluation,” Proceedings of EUROMICRO Conference on Digital System Design, pp.584-590, 2006

[75] S. Huda, M. Mallick, and J.H. Anderson, ”Clock Gating Architectures for FPGA Power Reduction,” Proceedings of IEEE International Conference on Field-Programmable Logic and Applications (FPL), pp. 112-118, Prague, Czech Republic, 2009.

[76] Q. Wang, S. Gupta, and J.H. Anderson, ”Clock Power Reduction for Virtex-5 FPGAs,” Proceedings of ACM/SIGDA International Conference on Field Programmable Gate Arrays (FPGA), pp. 13-22, Monterey, CA, February 2009.

[77] F. Rivoallon, ”Reducing Switching Power with Intelligent Clock Gating,” Xilinx WP370 (v1.1), July 2010.

[78] Actel, ”Dynamic Power Reduction in Flash FPGAs,” Application Note AC323, March 2009.

[79] M. Khan, ”Power Optimization in FPGA Designs,” Altera, 2006.

[80] Achronix, ”Introduction to Achronix FPGAs,” WP001 Rev.1.6, August 2008.

[81] Lattice Semiconductor, ”Practical Low Power CPLD Design,” August 2009.

[82] C.T. Chow, L.S.M. Tsui, P.H.W. Leong, W. Luk, and S. Wilton, ”Dynamic Voltage Scaling for Commercial FPGAs,” Proceedings of IEEE Int. Conf. on Field Prog. Technology, pp.215-222, 2005.

[83] C. Kao, ”Benefits of Partial Reconfiguration,” Xilinx, 2005.

[84] J. Becker, M. Hübner and M. Ullmann, ”Run-Time FPGA Reconfiguration for Power-/Cost-Optimized Real-time Systems,” Proceedings of International Conference on Very Large Scale Integration of System on Chip, pp. 119-132, December 2003.

[85] K. Paulsson, M. Hübner, S, Bayar, and J. Becker, ”Exploitation of Run-Time Partial Reconfiguration for Dynamic Power Management in Xilinx Spartan III-based Systems,” Proceedings of Reconfigurable Communication-centric SoCs, pp. 1-6, 2007.

[86] C. Jenkins and P. Ekas, ”Low-power Software-defined Radio Design using FPGAs,” Altera, 2006.

[87] A. Rahman, S. Das T. Tuan, and S. Trimberger, ”Determination of Power Gating Granularity for FPGA Fabric,” Proceedings of Custom Integrated Circuits Conference, pp. 9-12, Sept 2006.

[88] S. Ishihara, M. Hariyama, and M. Kameyama, ”A Low-power FPGA based on Autonomous Fine-grain Power-gating,” Proceedings of the

2009 Asia and South Pacific Design Automation Conference, pp. 119-120, 2009.

[89] P. S. Nair, S. Koppa, and E. B. John, ”A Comparative Analysis of Coarse-grain and Fine-grain Power Gating for FPGA Lookup Tables,” Proceedings of IEEE International Midwest Symposium on Circuits and Systems, pp.507-510, 2009

[90] A. Gayasen, Y. Tsai, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and T. Tuan, ”Reducing Leakage Energy in FPGAs using Region-constrained Placement,” Proceedings of the International Symposium on Field Programmable Gate Arrays, pp. 51-58, 2004.

[91] H. Hassan, M. Anis, A. El, and D.M. Elmasry, ”Activity Packing in FPGAs for Leakage Power Reduction,” Proceedings of Design, Automation and Test in Europe, pp.212-217, 2005.

[92] M. Khan, ”Power Optimization Innovations in 65-nm FPGAs,” Altera, March 2007.

[93] Atmel, ”Saving Power with Atmel PLDs,” 2000.

[94] Altera, ”Power Management in Portable Systems Using MAX II CPLDs,” Application Note 422, July 2006.

[95] Actel, ”Total System Power: Evaluating The Power Profile of FPGAs,” 2008.

[96] QuickLogic, ”Minimizing Energy Consumption with Very Low Power Mode in PolarPro Solution Platform CSSPs,” Application Note 88, 2008.

[97] S.H. Unger, ”Double-edge-triggered Flip-flops,” IEEE Trans. Computers, Vol. C-30, No. 6, pp. 447-451, June 1981.

[98] Xilinx, ”Low Power Design with CoolRunner-II CPLDs,” XAPP377, May 2002.

[99] R. Mukherjee and S. Ogrenci Memik, ”Evaluation of Dual VDD Fabrics for Low Power FPGAs,” Proceedings of Asia and South Pacific Design Automation Conference, pp. 1240-1243, January 2005.

[100] F. Li , Y. Lin , L. He , and J. Cong, ”Low-power FPGA using Pre-defined Dual-Vdd/Dual-Vt Fabrics,” Proceedings of ACM/SIGDA international

symposium on Field programmable gate arrays, pp. 42-50, February 2004.

[101] Y. Lin and L. He, ”Dual-Vdd Interconnect With Chip-Level Time Slack Allocation for FPGA Power Reduction,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 25, No. 10, pp. 2023-2034, Oct. 2006.

[102] F. Li, Y. Lin, and L. He, ”FPGA Power Reduction Using Configurable Dual-Vdd,” Proceedings of Design Automation Conference, pp.735-740, 2004.

[103] Y. Hu, Y. Lin, L. He, and T. Tuan, ”Physical Synthesis for FPGA Interconnect Power Reduction by Dual-Vdd Budgeting and Retiming,” ACM Transactions on Design Automation of Electronic Systems (TODAES), Vol. 13, No. 2, pp. 30:1-30:29, 2008.

[104] M. J. Alexander, ”Power Optimization for FPGA Look-up Tables,” Proceedings of International Symposium on Physical Design, pp. 156-162, 1997.

[105] G. Sutter, E. Todorovich, S. Lopez-Buedo, and E. Boemo, ”FSM Decomposition for Low Power in FPGA,” Proceedings of IEEE International Conference on Field-Programmable Logic and Applications (FPL), pp. 689-716, 2002.

[106] G. Sutter, E. Todorovich, S. Lopez-Buedo, and E. Boemo, ”Low-Power FSMs in FPGA: Encoding Alternatives,” Proceeding of Power and Timing Modeling, Optimization and Simulation, pp. 173-186, 2002.

[107] L. Mengibar, L. Entrena, M. G. Lorenz, and R. Sánchez-Reillo, ”State Encoding for Low-Power FSMs in FPGA,” Proceeding of Power and Timing Modeling, Optimization and Simulation, pp. 31-40, 2003.

[108] S. Douglass, ”Introducing the Virtex-5 FPGA Family,” Xilinx, 2006.

[109] N. Azizi and F.N. Najm, ”Look-up Table Leakage Reduction for FPGAs,” IEEE Custom Integrated Circuits Conference, pp. 186-189, 2005.

[110] C-F. Li, P-H. Yuh, C-L. Yang, and Y-W. Chang, ”Post-Placement Leakage Optimization for Partially Dynamic Reconfigurable FPGAs,” Proceedings of ACM/IEEE International Symposium on Low Power Electronics and Design, pp. 92-97, August 2007.

[111] J. H. Anderson, F. Najm, and T. Tuan, ”Active Leakage Power Optimization for FPGAs,” Proceedings of ACM/SIGDA International Symposium on Field programmable gate arrays, pp. 423-437,2004.

[112] H. Hassan, M. Anis, and M. Elmasry, ”Input Vector Reordering for Leakage Power Reduction in FPGAs,” IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems, Vol. 27, No. 9, pp. 1555-1564, Sept. 2008.

[113] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand, ”Leakage Current Mechanisms and Leakage Reduction Techniques in Deep-Submicrometer CMOS Circuits,” Proceedings of the IEEE, Vol. 91, No. 2, pp. 305-327, Feb. 2003.

[114] J.H. Anderson and F.N. Najm, ”A Novel Low-power FPGA Routing Switch,” Proceedings of IEEE Custom Integrated Circuits Conference, pp. 719-722, 2004.

[115] S. Mondal and S.O. Memik, ”A Low Power FPGA Routing Architecture,” Proceedings of IEEE International Symposium on Circuits and Systems, pp. 1222-1225, May 2005.

[116] F. Lie, Y. Lin, and L. He, ”Vdd Programmability to Reduce FPGA Interconnect Power,” Proceedings of the IEEE/ACM International Conference on Computer-aided Design, pp. 760-765, 2004.

[117] F.G. Wolff, M.J. Knieser, D.J. Weyer and C.A. Papachristou, ”High-level Low Power FPGA Design Methodology,” Proceedings of IEEE National Aerospace and Electronics Conference, pp. 554-559, 2000.

[118] D. Chen, J. Cong, and Y. Fan, ”Low-power High-level Synthesis for FPGA Architecture,” Proceedings of international symposium on Low Power Electronics and Design, pp. 134-139, 2003.

[119] K.O. Tinmaung, D. Howland, and R. Tessier, ”Power-aware FPGA Logic Synthesis using Binary Decision Diagrams,” Proceedings of the International Symposium on Field Programmable Gate Arrays, pp. 148-155, 2007.

[120] J.H. Anderson and F. N. Najm, ”Power-aware Technology Mapping for LUT-based FPGAs,” Proceedings of IEEE International Conference on Field-Programmable Technology, pp. 211-218, Dec 2002.

[121] D. Chen, J. Cong, F. Li, and L. He, ”Low-power Technology Mapping for FPGA architectures with dual supply voltages,” Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays, pp. 109-117, 2004.

[122] Z. Wang, E. Liu, J. Lai, and T. Wang, ”Power Minimization in LUT-based FPGA Technology Mapping,” Proceedings of Design Automation Conference, pp. 635-640, 2001.

[123] A. H. Farrahi and M. Sarrafzadeh, ”FPGA Technology Mapping for Power Minimization,” Proceedings of International Workshop on Field-Programmable Logic and Applications: Field-Programmable Logic, Architectures, Synthesis and Applications, pp. 66-77, 1994.

[124] H. Li, W. Mak, and S. Katkoori, ”Efficient LUT-based FPGA Technology Mapping for Power Minimization,” Proceedings of Asia and South Pacific Design Automation Conference, pp. 353-358, 2003.

[125] C. Chen, T. Hwang, and C. L. Liu, ”Low Power FPGA Design - a Re-engineering approach,” Proceedings of Design Automation Conference, pp. 656-661, 1997.

[126] B. Kumthekar, L. Benini, E. Macii, and F. Somenzi, ”Power Optimization of FPGA-based Designs without Rewiring,” IEEE Proceedings Computers and Digital Techniques, Vol. 147, No. 3, pp. 167-174, May 2000.

[127] F. Li, D. Chen, L. He, and J. Cong, ”Architecture Evaluation for Power-efficient FPGAs,” Proceedings of International Symposium on Field Programmable Gate Arrays, pp. 175-184, Feb. 2003.

[128] A. Singh and M. Marek-Sadowska, ”Efficient Circuit Clustering for Area and Power Reduction in FPGAs,” Proceedings of International Symposium on Field Programmable Gate Arrays, pp. 59-66, Feb. 2002.

[129] D. Chen and J. Cong, ”Delay Optimal Low-power Circuit Clustering for FPGAs with Dual Supply Voltages,” Proceedings of International Symposium on Low Power Electronics and Design, pp. 70-73, 2004.

[130] S. Gupta, J.H. Anderson, L. Farragher and Q. Wang, ”CAD Techniques for Power Optimization in Virtex-5 FPGAs,” Proceedings of IEEE Custom Integrated Circuits Conference, pp. 85-88, 2007.

[131] K. Vorwerk, M. Raman, J. Dunoyer, Y. Hsu, A. Kundu, and A. Kennings, ”A Technique for Minimizing Power During FPGA Placement,” Proceedings of International Conference on Field Programmable Logic and Applications, pp. 233-238, Sept. 2008.

[132] J. Lamoureux and S.J.E. Wilton, ”Clock-aware Placement for FPGAs,” Proceedings of International Conference on Field Programmable Logic and Applications, pp. 124-131, Aug. 2007.

[133] J. Lamoureux and S.J.E. Wilton, ”On the Interaction between Power-Aware CAD Algorithms for FPGAs,” Proceedings of IEEE International Conference on Computer Aided Design, pp. 701-708, 2003.

[134] E. P. Ramo, J Resano, D. Mozos, and F. Catthoor, ”A Configuration Memory Hierarchy for Fast Reconfiguration with Reduced Energy Consumption Overhead,” Proceedings of IEEE 13th Reconfigurable Architectures Workshop, pp. 178, 2006.

[135] H. Wang, M. Miranda, A. Papaniko, F. Catthoor, and W. Dehaene, ”Variable Tapered Pareto Buffer Design and Implementation Allowing Run-Time Configuration for Low Power Embedded SRAMs,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 13, No. 10, pp. 1127-1135, 2005.

[136] D. Howland and R. Tessier, ”RTL Dynamic Power Optimization for FPGAs,” Proceedings of IEEE Midwest Symp. on Circuits and Systems, pp. 714-717, 2008.

[137] J.H. Anderson and C. Ravishankar, ”FPGA Power Reduction by Guarded Evaluation,” Proceedings of the International Symposium on Field Programmable Gate Arrays, pp. 157-166, 2010.

[138] Y. Meng, T. Sherwood, and R. Kastner, ”Leakage Power Reduction of Embedded Memories on FPGAs through Location Assignment,” Proceedings of Design Automation Conference, pp. 612-617, 2006.

[139] L.Wang, M. French, A. Davoodi, and D. Agarwal, ”FPGA Dynamic Power Minimization through Placement and Routing Constraints,” EURASIP J. Embedded Syst., Vol. 2006, No. 1, pp. 7-17, 2006.

[140] C. Chen, R. Parsa, N. Patil, S. Chong, K. Akarvardar, J. Provine, D. Lewis, J. Watt, R.T. Howe, H.-S. P. Wong, and S. Mitra, ”Efficient FPGAs using Nanoelectromechanical Relays,” Proceedings of the Interna-

tional Symposium on Field Programmable Gate Arrays, pp. 273-282, 2010.

[141] A. Gayasen, S. Srinivasan, N, Vijaykrishnan, and M. Kandemir, ”Design of Power-aware FPGA Fabrics,” International Journal of Embedded Systems, Vol. 3, Nos. 1/2, pp.52-64, 2007.

[142] V. George, H. Zhang, and J. Rabaey, ”The Design of a Low Energy FPGA,” Proceedings of the International Symposium on Low Power Electronics and Design, pp. 188-193, 1999.

[143] Y. Matsumoto and A. Masaki, ”Low-Power FPGA using Partially Low Swing Routing Architecture,” Electronics and Communications in Japan, Part 3, Vol. 88, No. 11, pp. 11-19, 2005.

[144] T. Wu and Y. Lin, ”Storage Optimization by Replacing Some Flip-Flops with Latches,” Proceedings of the Conference on European Design Automation, pp. 296-301, 1996.

[145] D. Chinnery, K. Keutzer, J. Sanghavi, E. Killian, and K. Sheth, ”Automatic Replacement of Flip-Flops by Latches in ASICs,” Closing the Gap Between ASIC and Custom, chapter 7, pp. 187-208, 2002.

[146] T. Pi and P.J. Crotty, ”FPGA Lookup Table with Transmission Gate Structure for Reliable Low-Voltage Operation,” U.S. Patent 6667635, Dec. 2003.

[147] University of California Berkeley, ”Berkeley Logic Interchange Format (BLIF),” February 2005.

[148] T. Becker, W. Luk, and P.Y.K. Cheung, ”Enhancing Relocatability of Partial Bitstreams for Run-Time Reconfiguration,” Proceedings of Symposium on Field-Programmable Custom Computing Machines, pp. 35-44, 2007.

[149] H. Kalte, G. Lee, M. Porrmann, and U. Rückert, ”REPLICA: A Bitstream Manipulation Filter for Module Relocation in Partial Reconfigurable Systems,” Proceedings of International Parallel and Distributed Processing Symposium, pp.151b, 2005.

[150] H. Kalte and M. Porrmann, ”REPLICA2Pro: Task Relocation by Bitstream Manipulation in Virtex-II/Pro FPGAs,” Proceedings of International Conference on Computing Frontiers, pp. 403-412, 2006.

[151] Edson L. Horta and John W. Lockwood, ”Automated Method to Generate Bitstream Intellectual Property Cores for Virtex FPGAs,” Proceedings of International Conference on Field-Programmable Logic and Applications, pp. 975-979, 2004.

[152] Y. E. Krasteva, A. B. Jimeno, E. Torre, and T. Riesgo, ”Straight Method for Reallocation of Complex Cores by Dynamic Reconfiguration in Virtex II FPGAs,” Proceedings of International Workshop on Rapid System Prototyping, pp. 77-83, 2005.

[153] http://www.fpga-faq.org/

[154] S.P. Young and T.J. Bauer, ”Architecture and Method for Partially Reconfiguring an FPGA,” US Patent No. 6526557 B1, 2003.

[155] P. Hallschmid and S.J.E. Wilton, ”Detailed Routing Architectures for Embedded Programmable Logic IP Cores,” Proceedings of International Symposium on Field Programmable Gate Arrays, pp. 69-74, 2001.

[156] Xilinx, Inc, ”Virtex-II Platform FPGA User Guide,” Xilinx user guide UG002, 2007.

[157] S. Hauck, ”Configuration Prefetch for Single Context Reconfigurable Coprocessors,” Proceedings of International Symposium on Field-Programmable Gate Arrays, pp. 65-74, 1998.

[158] B. Jeong, S Yoo, S. Lee, and K. Choi, ”Hardwaresoftware Cosynthesis for Run-time Incrementally Reconfigurable FPGAs,” Proceedings of Asia South-Pacific Design Automation Conference, pp. 169-174, Jan. 2000.

[159] H. Schmit, ”Incremental Reconfiguration for Pipelined Applications,” Proceedings of Symposium on FPGAs for Custom Computing Machines, pp. 47-55, 1997.

[160] W. Luk, N. Shirazi, S.R. Guo, and P.Y.K. Cheung, ”Pipeline Morphing and Virtual Pipelines,” Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications, pp. 111-120, 1997.

[161] Z. Li, K. Compton, and S. Hauck, ”Configuration Caching Techniques for FPGA,” Proceedings of Symposium on FPGA for Custom Computing Machines, pp. 22-38, 2000.

[162] S. Trimberger, ”A Time-Multiplexed FPGA,” Proceedings of Symposium on FPGA-Based Custom Computing Machines, pp. 22-28, 1997.

[163] A. DeHon, ”Dynamically Programmable Gate Arrays: A Step Toward Increased Computational Density,” Proceedings of the Canadian Workshop on Field-Programmable Devices, pp. 47-54, 1996.

[164] R. A. Stefan and S. D. Cotofana, ”Bitstream Compression Techniques for Virtex 4 FPGAs,” Proceedings of International Conference on Field Programmable Logic and Applications, pp. 323-328, September 2008.

[165] S. Hauck, Z. Li, and E. Schwabe, ”Configuration Compression for the Xilinx XC6200 FPGA,” Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, pp. 138-146, 1998.

[166] Z. Li and S. Hauck, ”Dont Care Discovery for FPGA Configuration Compression,” Proceedings of the International Symposium on Field Programmable Gate Arrays, pp. 91-98, 1999.

[167] J.H. Pan, T. Mitra, and W-F. Wong, ”Configuration bitstream compression for dynamically reconfigurable FPGAs,” Proceedings of International Conference on Computer Aided Design (ICCAD), pp. 766-773, November 2004.

[168] Z. Li and S. Hauck, ”Configuration Compression for Virtex FPGAs,” Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 147-159, 2001.

[169] S. Hauck and W.D. Wilson, ”Runlength Compression Techniques for FPGA Configurations,” Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, pp. 286-287, 1999.

[170] A. Dandalis and V. K. Prasanna, ”Configuration Compression for FPGA-based Embedded Systems,” Proceedings of the International Symposium on Field Programmable Gate Arrays, pp. 173-182, 2001.

[171] U. Malik and O. Diessel, ”A Configuration Memory Architecture for Fast Run-time Reconfiguration of FPGAs,” Proceedings of International Conference on Field Programmable Logic and Applications, pp. 636-639, 2005.

[172] A. Jara-Berrocal and A. Gordon-Ross, ”Runtime Temporal Partitioning Assembly to Reduce FPGA Reconfiguration Time,” Proceedings of International Conference on Reconfigurable Computing and FPGAs, pp. 374-379, December 2009.

[173] U. Malik and O. Diessel, ”On the Placement and Granularity of FPGA Configurations,” Proceedings of the International Conference on Field-Programmable Technology (FPT), pp. 161-168, 2004.

[174] J. T. Young, J. V. Lindholm, and I. L. McEwen, ”Routing with Frame Awareness to Minimize Device Programming Time and Test Cost,” US Patent No. 7149997 B1, 2006.

[175] J. Resano, D. Mozos, D. Verkest, S. Vernalde, and F. Catthoor, ”Run-time Minimization of Reconfiguration Overhead in Dynamically Reconfigurable Systems,” Proceedings of International Conference on Field-Programmable Logic and Applications, pp. 585-594, 2003.

[176] S. Koh and O. Diessel, ”Module Graph Merging and Placement to Reduce Reconfiguration Overheads in Paged FPGA Devices,” Proceedings of International Conference on Field-Programmable Logic and Applications, pp. 293-298, 2007.

[177] N. Shirazi , W. Luk , and P. Y. K. Cheung, ”Automating Production of Run-Time Reconfigurable Designs,” Proceedings of Symposium on FPGAs for Custom Computing Machines, pp. 147-156, April 15-17, 1998.

[178] K. P. Raghuraman, H. Wang, and S. Tragoudas, ”A Novel Approach to Minimizing Reconfiguration Cost for LUT-Based FPGAs,” Proceedings of International Conference on VLSI Design, pp. 673-676, 2005.

[179] H. Tan and R. F. DeMara, ”A Physical Resource Management Approach to Minimizing FPGA Partial Reconfiguration Overhead,” Proceedings of International Conference on Reconfigurable Computing and FPGAs, pp. 86-90, 2006.

[180] V. Rana, S. Murali, D. Atienza, M. Santambrogio, L. Benini, and D. Sciuto, ”Minimization of the Reconfiguration Latency for the Mapping of Applications on FPGA-based Systems,” Proceedings of International Conference on Hardware/software Codesign and System Synthesis, pp. 325-334, 2009.

[181] T. Degryse, K. Bruneel, H. Devos, and D. Stroobandt, ”Loop Transformations to Reduce the Dynamic FPGA Reconfiguration Overhead,” Proceedings of International Conference on Reconfigurable Computing and FPGAs, pp. 133-138, 2008.

[182] K. Bondalapati and V. K. Prasanna, ”Loop Pipelining and Optimization for Run Time Reconfiguration,” Proceedings of the IPDPS Workshops on Parallel and Distributed Processing, pp. 906-915, May 2000.

[183] S. Ghiasi, A Nahapetian, and M Sarrafzadeh, ”An Optimal Algorithm for Minimizing Run-time Reconfiguration Delay,” ACM Transactions on Embedded Computing Systems (TECS), Vol. 3, No. 2, pp. 237-256, May 2004.

[184] R. Kalra and R. Lysecky, ”Configuration Locking and Schedulability Estimation for Reduced Reconfiguration Overheads of Reconfigurable Systems,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 18, No. 4, pp. 671-674, April 2010.

[185] W. Chen, Y. Wang, X. Wang, and C. Peng, ”A New Placement Approach to Minimizing FPGA Reconfiguration Data,” Proceedings of International Conference on Embedded Software and Systems, pp. 169-174, 2008.

[186] F. Mehdipour, M. S. Zamani, H. R. Ahmadifar, M. Sedighi, and K. Murakami, ”Reducing Reconfiguration Time of Reconfigurable Computing Systems in Integrated Temporal Partitioning and Physical Design Framework,” Proceedings of IEEE 20th Int. Parallel Distrib. Process. Symp. (IPDPS), pp. 25-29, 2006.

[187] H. Kalte, M. Porrmann, and U. Rckert, ”Study on Column Wise Design Compaction for Reconfigurable Systems,” Proceedings of International Conference on Field Programmable Technology, pp. 413-416, Dec. 2004.

[188] R. Hartenstein, ”Coarse Grain Reconfigurable Architectures”, Proceedings of the Asia and South Pacific Design Automation Conference, pp. 564-570, 2001.

[189] S. Shukla, N. W. Bergmann, and J. Becker, ”QUKU: A Two-Level Reconfigurable Architecture,” Proceedings of IEEE Computer Society Annual Symposium on VLSI: Emerging VLSI Technologies and Architectures, pp. 109-116, 2006.

[190] J. Delorme, A. Nafkha, P. Leray, and C. Moy, ”New OPBHWICAP Interface for Realtime Partial Reconfiguration of FPGA,” Proceedings of the International Conference on Reconfigurable Computing and FPGAs, pp. 386-391, Dec. 2009.

[191] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, ”Run-time Partial Reconfiguration Speed Investigation and Architectural Design Space Exploration,” Proceedings of the International Conference on Field Programmable Logic and Applications, pp. 498-502, Aug. 2009.

[192] C. Claus, B. Zhang,W. Stechele, L. Braun, M. Huebner, and J. Becker, ”A Multiplatform Controller Allowing for Maximum Dynamic Partial Reconfiguration Throughput,” Proceedings of the International Conference on Field Programmable Logic and Applications, pp. 535-538, Sept. 2008.

[193] E. Moscu Panainte, K.L.M. Bertels, and S. Vassiliadis, ”Compiler-driven FPGA-area Allocation for Reconfigurable Computing,” Proceedings of Design, Automation and Test in Europe, pp. 369-374, March 2006.

[194] N. Moreano, E. Borin, C. de Souza, and G. Araujo, ”Efficient Datapath Merging for Partially Reconfigurable Architectures,” IEEE Transaction on Computer Aided Design, Vol. 24. No.7, pp. 969-980, 2005.

[195] S. Lange and M. Middendorf, ”Hyperreconfigurable Architectures for Fast Runtime Reconfiguration,” Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines, pp.304-305, 2004.

[196] J.Y.-C. Kuo, H. ElGindy, A.K.-A. Ku, ”A Novel Network Architecture Support for Fast Reconfiguration,” Proceedings of the International Conference on Field-Programmable Technology, pp. 353-356, Dec. 2007.

[197] J.Y.-C. Kuo, A.K.-A. Ku, J. Xue, O. Diessel, U. Malik, ”ACS: an Addressless Configuration Support for Efficient Partial Reconfigurations,” Proceedings of the International Conference on Field-Programmable Technology, pp. 161-168, Dec. 2008.

[198] Xilinx, Inc., ”‘Xilinx ISE 8 In-Depth Tutorial,” August 2006.

[199] Xilinx, Inc, ”Early Access Partial Reconfiguration User Guide,” Xilinx user guide UG208, 2006.

[200] X. Wu and M. Pedram, ”Low-power Sequential Circuit Design Using T Flip-flops,” International Journal of Electronics, Vol. 88, No. 6, pp. 635-643, June 2001.

[201] D. Markovic, B. Nikolic, and R. Brodersen, ”Analysis and Design of Low-energy Flip-flops,” Proceedings of International Symposium on Low Power Electronics and Design, pp.52-55, August 2001.

[202] C. Rossmeissl, A. Sreeramareddy, and A. Akoglu, ”Partial Bitstream 2-D Core Relocation for Reconfigurable Architectures,” Proceedings of NASA/ESA Conference on Adaptive Hardware and Systems, pp.98-105, 2009.

[203] S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. M. Panainte, ”The Molen Polymorphic Processor,” IEEE Transactions on Computers, pp. 1363-1375, November 2004.

List of Publications

International Conference Proceedings

1. T. Marconi, D. Theodoropoulos, K.L.M. Bertels, G. N. Gaydadjiev, **A Novel HDL Coding Style to Reduce Power Consumption for Reconfigurable Devices**, *Proceedings of the International Conference on Field-Programmable Technology (FPT)*, Beijing, China, pp. 295-299, December 2010.
2. T. Marconi, J.Y. Hur, K.L.M. Bertels, G. N. Gaydadjiev, **A Novel Configuration Circuit Architecture to Speedup Reconfiguration and Relocation for Partially Reconfigurable Devices**, *Proceedings of IEEE Symposium on Application Specific Processors (SASP)*, Anaheim, CA, USA, pp. 105-110, June 2010.
3. T. Marconi, Y. Lu, K.L.M. Bertels, G. N. Gaydadjiev, **3D Compaction: a Novel Blocking-aware Algorithm for Online Hardware Task Scheduling and Placement on 2D Partially Reconfigurable Devices**, *Proceedings of the International Symposium on Applied Reconfigurable Computing (ARC)*, Bangkok, Thailand, pp. 194-206, March 2010.
4. T. Marconi, Y. Lu, K.L.M. Bertels, G. N. Gaydadjiev, **A Novel Fast Online Placement Algorithm on 2D Partially Reconfigurable Devices**, *Proceedings of the International Conference on Field-Programmable Technology (FPT)*, Sydney, Australia, pp. 296-299, December 2009.
5. T. Marconi, Y. Lu, K.L.M. Bertels, G. N. Gaydadjiev, **Intelligent Merging Online Task Placement Algorithm for Partial Reconfigurable Systems**, *Proceedings of Design, Automation and Test in Europe (DATE)*, Munich, Germany, pp. 1346-1351, March 2008.
6. T. Marconi, Y. Lu, K.L.M. Bertels, G. N. Gaydadjiev, **Online Hardware Task Scheduling and Placement Algorithm on Partially Reconfigurable Devices**, *Proceedings of International Workshop on Applied Reconfigurable Computing (ARC)*, London, UK, pp. 306-311, March 2008.

Technical Reports

1. T. Marconi, K.L.M. Bertels, G. N. Gaydadjiev, **A Novel Logic Element for Power Reduction in FPDs**, *CE-TR-2010-01*, Computer Engineering Lab, TU Delft, January 2010.
2. T. Marconi, D. Theodoropoulos, K.L.M. Bertels, G. N. Gaydadjiev, **A Novel HDL Coding Style for Power Reduction in FPGAs**, *CE-TR-2010-02*, Computer Engineering Lab, TU Delft, January 2010.

Other publications, not directly related to this dissertation:

International Conference Proceedings

1. Y. Lu, T. Marconi, K.L.M. Bertels, G. N. Gaydadjiev, **A Communication Aware Online Task Scheduling Algorithm for FPGA-based Partially Reconfigurable Systems**, *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines(FCCM)*, Charlotte, North Carolina, USA, pp. 65-68, May 2010.
2. Z. Nawaz, T. Marconi, T. P. Stefanov, K.L.M. Bertels, **Flexible Pipelining Design for Recursive Variable Expansion**, *Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Rome, Italy, pp. 1-8, May 2009.
3. Y. Lu, T. Marconi, K.L.M. Bertels, G. N. Gaydadjiev, **Online Task Scheduling for the FPGA-Based Partially Reconfigurable Systems**, *Proceedings of International Workshop on Applied Reconfigurable Computing (ARC)*, Karlsruhe, Germany, pp. 216-230, March 2009.
4. Y. Lu, T. Marconi, G. N. Gaydadjiev, K.L.M. Bertels, R. J. Meeuws, **A Self-adaptive on-line Task Placement Algorithm for Partially Reconfigurable Systems**, *Proceedings of the 22nd Annual International Parallel and Distributed Processing Symposium (IPDPS) - RAW*, Miami, Florida, USA, pp. 1-8, April 2008.
5. Y. Lu, T. Marconi, G. N. Gaydadjiev, K.L.M. Bertels, **An Efficient Algorithm for Free Resources Management on the FPGA**, *Proceedings of Design, Automation and Test in Europe (DATE)*, Munich, Germany, pp. 1095-1098, March 2008.

6. Z. Nawaz, O.S. Dragomir, T. Marconi, E. Moscu Panainte, K.L.M. Bertels, S. Vassiliadis, **Recursive Variable Expansion: A Loop Transformation for Reconfigurable Systems**, *Proceedings of International Conference on Field-Programmable Technology(FPT)*, Kokurakita, Kitakyushu, Japan, pp. 301-304, December 2007.

Local Conference Proceedings

1. Z. Nawaz, T. Marconi, T. P. Stefanov, K.L.M. Bertels, **Optimal pipeline design for Recursive Variable Expansion**, *Proceedings of Fifth International Summer School on Advanced Computer Architecture and Compilation for Embedded Systems (ACACES)*, Terrassa, Spain, pp. 85-88, July 2009.
2. Y. Lu, T. Marconi, G. N. Gaydadjiev, K.L.M. Bertels, **A new model of placement quality measurement for online task placement**, *Proceedings Workshop on Circuits, Systems and Signal Processing (ProRISC)*, Veldhoven, The Netherlands, pp. 307-310, November 2007.

Samenvatting

Het aanwenden van partial reconfigurable devices voor runtime reconfigurable systems kan een vermindering bewerkstelligen van hardware area, energieverbruik, economische kosten, bitstream grootte en herconfiguratie tijd in aanvulling op performance verbeteringen, die te danken zijn aan beter maatwerk. Maar om deze voordelen te verkrijgen, dienen de gebruikers bijkomende kosten te betalen: hoger stroomverbruik, meer silicon area en lagere verwerkingssnelheden in vergelijking met ASIC's. Hoger stroomverbruik vereist hogere verpakkingskosten, verkort de chip-levensduur, vereist dure koelsystemen, vermindert de betrouwbaarheid van het systeem en belet het gebruik van een batterij. Om de minder efficiënte gebruik van de ruimte op de FPGA tegen te gaan, moet men runtime reconfigureren en de posities van draaiende taken herorganiseren. Aangezien de beschikbare configuratie-datapaden gebruikelijk een gelimiteerde bandbreedte hebben, kan de hoge overhead van de herconfiguratie de voordelen van een dynamisch systeem te niet doen. In dit proefschrift richten we ons op drie belangrijke problemen om deze voordelen meer toe te passen. Om precies te zijn, zijn dat: de online inroostering en plaatsing van hardware taken, vermindering van het stroomverbruik, het terugschroeven van de runtime reconfiguration overhead. Aangezien hardware taken dynamisch worden toegewezen en vrijgegeven tijdens de executie van het systeem, kan de reconfigurable fabric lijden aan fragmentatie. Dit kan leiden tot de ongewenste situatie dat de taken niet kunnen worden toegewezen, zelfs als er voldoende vrije ruimte beschikbaar zou zijn. Als gevolg worden de algehele prestaties van het systeem aangetast. Dus efficiënt beheer van de resources in hardware is erg belangrijk. Om hardware resources efficiënt te beheren, stellen we vernieuwende online hardware taak inroosterings- en plaatsingsalgoritmen voor op partial reconfigurable devices. Deze algoritmen zijn zowel sneller als van hogere kwaliteit dan bestaande aanpakken. Met het oog op het reduceren van hoog stroomverbruik in FPD's stellen we een nieuw Logic Element (LE) voor met een lager stroomverbruik dan huidige FPD's. Om runtime overhead te reduceren stellen we een nieuwe FPGA configuratie-circuit architectuur voor met snellere herconfiguratie en herplaatsing in vergelijking met huidige FPGA's.

Curriculum Vitae

Thomas Marconi was born on the 30th of April 1973 in Belinyu, Bangka, Indonesia. In 1996, he graduated from a 5 year university degree in Electrical Engineering and got his Electrical Engineer Degree with ”Cum Laude” from Krida Wacana Christian University, Jakarta, Indonesia. During his first degree study, he worked as a teaching assistant and a lab assistant in the same university. After successfully defending his Electrical Engineer’s thesis in 1995, he started working as a Service Engineer at STIMEC ELCOM P.T., Jakarta, Indonesia.

After working in industry, in 1996, he decided to come back to campus and started working as the youngest permanent faculty member at Electrical Engineering Department of Tarumanagara University, Jakarta, Indonesia. To learn more about Electrical Engineering, he studied and did his research at Bandung Institute of Technology (ITB), Bandung, Indonesia to pursue his Master of Electrical Engineering Degree with a full scholarship from Tarumanagara Foundation during 1998-2000. After successfully defending his master thesis, in 2000, he obtained his Master of Electrical Engineering with ”Cum Laude” from Microelectronic Lab of ITB. During his study at ITB, he was the best master engineering graduated student. After obtaining his master degree, he was promoted as the youngest ”LEKTOR” (equivalent to Assistant Professor) in Electrical Engineering Department of Tarumanagara University. During his academic career, he has taught several subjects in Electrical Engineering and supervised some theses of Electrical Engineers.

In 2006, he joined the Computer Engineering Lab of TU Delft in the Netherlands as a PhD researcher under the guidance of Prof. Dr. Stamatis Vassiliadis, Dr. Georgi Gaydadjiev, and Dr. Koen Bertels. During his work at TU Delft, he participated in a European Commission funded project called hArtes (Holistic Approach to Reconfigurable Real Time Embedded Systems).

His research interests include: reconfigurable computing, online hardware task scheduling and placement, low power designs, runtime reconfigurable systems, computer architecture, and microprocessor designs. He has been serving as an external reviewer for many conferences and journals, e.g., DATE, MICRO, FPL, SAMOS, HiPC, ARC, HiPEAC, VLSI-SOC and more. He is an affiliate member of European Network of Excellence HiPEAC.

Runtime reconfigurable systems built upon devices with partial reconfiguration can provide reduction in overall hardware area, power efficiency, and economic cost in addition to the performance improvements due to better customization. However, the users of such systems have to be able to accept the additional costs compared to the hardwired application specific circuits. More precisely reconfigurable devices have higher power consumption, occupy larger silicon area and operate at lower speeds. Higher power consumption requires additional packaging cost, shortens chip lifetimes, requires expensive cooling systems, decreases system reliability and prohibits battery operation. The less efficient usage of silicon real estate is usually compensated by the runtime hardware reconfiguration and functional units relocation. The available configuration data paths, however, have limited bandwidth that introduces overheads that may eclipse the benefits of dynamic reconfiguration. In this dissertation, we address three major problems related to hardware resources runtime management: efficient online hardware task scheduling and placement, power consumption reduction and reconfiguration overhead minimization. Since hardware tasks are allocated and deallocated dynamically at runtime, the reconfigurable fabric can suffer of fragmentation. This can lead to situations when tasks cannot be allocated even if there sufficient free area is available. As a result, the overall system performance is degraded. Therefore, efficient hardware management of resources is very important. To manage hardware resources efficiently, we propose novel online hardware task scheduling and placement algorithms on partially reconfigurable devices with higher quality and faster execution compared to related proposals. To cope with the high power consumption in field programmable devices, we propose a novel logic element with lower power consumption compared to current approaches. To reduce runtime overhead, we augment the FPGA configuration circuit architecture and allow faster reconfiguration and relocation compared to current reconfigurable devices.

CE
TUDelft
Delft University of Technology
ISBN 978-90-72298-20-1

EFFICIENT RUNTIME MANAGEMENT OF RECONFIGURABLE HARDWARE RESOURCES

THOMAS MARCONI