Register Manipulation using Automatic Register Balancing

A
M.Tech Phase I Report
Submitted in Partial Fulfillment of the Requirements
for the Degree
of

MASTER OF TECHNOLOGY

by

Rajesh Kumar Jha
(174101007)

under the guidance of

Dr. Chandan Karfa

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
GUWAHATI - 781039, ASSAM

CERTIFICATE

*This is to certify that the work contained in this thesis entitled "**Register Manipulation using Automatic Register Balancing** " is a bonafide work of **Rajesh Kumar Jha (Roll No. 174101007)**, carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati under my supervision and that it has not been submitted elsewhere for a degree.*

Supervisor: **Dr. Chandan Karfa**

Assistant Professor,

Department of Computer Science & Engineering,

Indian Institute of Technology Guwahati, Assam.

November, 2018

Guwahati.

Contents

List of Figures

Abstract

Digital circuits are backbone of any device being used in modern era, and so critical is their design. A specific design is effected by many factors and many features depend on how the design is being realised. Latency of a circuit is one of them. Sometimes we may want to increase the latency of a given circuit and sometimes we may want to reduce the same. In most of cases we want to reduce it. But manually doing the same is an uphill task.

The work aims to present a method to automate the entire process using standard global retiming theorem. Moreover developers can manually add, remove and move register to locations of their choice. To help the developers a python implementation of the same has also been provided.

Chapter 1

Introduction

1.1 Domain of Research

Many a times its been seen that developers face issues with register balancing during the formation of complex hierarchical design. At the time of performance fine tuning some path or most of the paths may not able to meet the clock period that is desired. So naturally they think of inserting registers in those paths in order to achieve the desired time period. Moreover the designers may want to shift one or more registers from a given location to some other location or may remove registers form a location.

In order to keep the functionality intact adding a register isnt enough, we have to add registers to all the parallel paths. Now the real problem arises when a path has thousands of parallel paths. None of present day high level tools support automatic register balancing.

1.2 Why the problem is complex

The problem on a small scale seems feasible but for large scale it is an uphill task to perform. Moreover to add a single register we have to balance out the same on rest of all the parallel paths. And there can me millions of parallel paths. Moreover a small mistake or imbalanced circuits can totally alter the functionality than the desired. Apart from that, the state of art technique for balancing the same is not efficient enough to handle very large circuits The method proposed here will automatically balance all the registers in the corresponding parallel paths when a register manipulation is performed. The register balancing problem in digital circuits has been suitably mapped to the standard global retiming algorithm. The methods works on any design size preserving the core functionalities of the circuit.

1.3 Previous Work

The main work on this topic has been contributed by [1]. The work presented here is a continuation of the idea presented in [1]. The results found in [1] state that the problem of register balancing can be smoothly solved by the global retiming algorithm. This thesis presents a robust implementation of the idea portrayed by [1] with added functionalities like addition, deletion and moving registers. The previous work is extended using the same principles of global retiming theorem and more features has been added to it. Moreover this work implements the work done in [1] with improved complexities. Figure 1 shows a comparison between the previous and new time complexities based on standard benchmark circuits. Green line shows old complexities and the blue line depicts the new complexities and the time is measured in seconds.

1.4 Motivation and Application

Manipulation of registers in a digital circuit serves a crucial role in every possible sense. Be it latency or frequency, registers provide a handy way to play with these properties. But adding or removing registers is not as easy as its said. In todays world, where time is considered to be

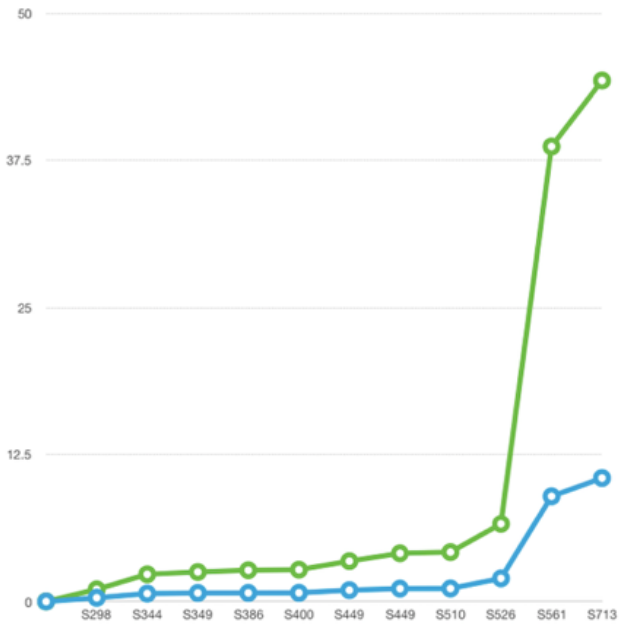

Fig. 1.1 Regular vs Optimized retiming algorithm

the most precious resource, no one is willing to invest time to balance registers in a given digital circuits. So it is important to develop a tool so that the same can be automated. The thesis works in the same direction so as to make the lives of developers easy. The same can be used by model based High level synthesis tool users. This can be used by them during performance fine tuning.

1.5 Organization of Thesis

The thesis has been organized as follows. Chapter 2 gives a crisp literature survey of the topic under light namely standard global retiming algorithm for clock period minimization. Chapter 3 revolves around the work done and the implementation proposed. It shows the complexities of the work and the logics involved. Chapter 4 here draws the conclusion of the above said work and supplements with the future work.

Chapter 2

Background / Framework

2.1 Global retiming for clock period minimization

Retiming can be defined as a transformation which in turn brings a change in the position of sequential elements present in the said design without any alteration in the input-output features of the circuit.

We can represent any given sequential circuit using a graph which is directed G(V, E), where each of the vertex v V corresponds to the given design unit. Abstraction level decides the design unit and we can have gates for the gate level design and we can have functional units etc for the given high level design. Each given edge portrays the flow of signal from the input of the edge to the input. In the same fashion each edge weight represents the number of registers in that edge and each of them are non-negative. Each vertex possesses a constant computational delay and let us assume the delay of node v is given by d(v) and it cannot be negative. Retiming aims to map the design provided G to a retimed design Gr. A retiming can be termed as labelling of the vertices each vertices to a set of integers. The weight of the retimed design can be given by :

$$w_r(e_{u,v}) = r(v) + w(e_{u,v}) - r(u)$$

The retiming label originally depicts the motion of registers form the output towards its input. So, the task of retiming is to allocate retiming labels to all the combinational blocks of

the design. Let p be a path, it can be defined as a sequence of alternating edges and vertices in a fashion that all the vertices are successive, that is fan out of the last one. The weight of a path p can be defined as the summation of all the weights of the edges we find in the path. And same goes with the computational delay, summation of all the computational delays of all the vertices that fall in the path. So, the clock period of a path p can be found out by :

$$c = \max_{p|w(p)=0} \{d(p)\}$$

The goal we have here is to minimize the clock period c. Let us define two terms important for the formulation W(u, v) and D(u, v). The first term specifies the minimum registers on any path from the first node to the second node, the later denotes the maximum computational delay incurred summing all the paths from the first node to the second node with the given weight. The matrix W can be populated using algorithms for all pair shortest path like Floyd Warshall since edge weights are zero or positive and we need to get the shortest path among each vertex pairs. Computation of D is not same as computation of W since each entry in D matrix depends on the corresponding value of W. There exists a unified method to populate both the matrices simultaneously. Now the constraints of global retiming for a specific target clock period c can be formulated using a set of two inequalities.

- (feasibility constraint): The edge-weight must be nonnegative after retiming. Formally, $w_r(e_{u.v}) \geq 0, \forall e_{u.v} \in E$. Using equation 1, $r(u) - r(v) \leq w(e_{u.v}), \forall e_{u.v} \in p$ '.

- (critical path constraints:) The computational delay of 0-weight paths after retiming must be less than or equal to c. If $D(u, v) > c$, then $r(v) - r(u) + w(u, v) \geq 1$ must hold for the critical path to have computation time less than or equal to c. Formally, $r(u) - r(v) \leq w(u, v) - 1$ for all paths from u to v with $D(u, v) > c$.

The aim here is to get the values of r namely retiming labels for all vertices satisfying the stated set of constraints. Now the values can be constructed using a graph called constraint graph which can be generated using the above set of inequalities and solving the same using an all pair shortest path algorithm like bellman ford.

Chapter 3

Proposed Solution

3.1 Automatic Register Balancing

Register balancing automatically balances all the parallel paths when a register is inserted in a given path. When appropriate delays are assigned to nodes of a circuit, then the register balancing problem can be mapped to global retiming problem. Specifically, we tend to produce appropriate retiming model of the circuit considering the register balancing constraints so apply existing retiming algorithm to retime the circuit. And within the retimed circuit, registers are inserted and parallel ways are balanced. The work done in [1] shows how we can balance registers using standard global retiming. This work builds on the same principle. For addition we add a node after the said node and increase its latency to maximum latency possible. So the software puts a register in between to break the critical path. Same goes for deletion, we create a deficiency of registers in front so all registers are move from that location to the front.

3.2 Adding Registers

Sometimes we can explicitly want to add some registers in our circuits. It can have various reasons. Automatic register balancers do not let us do manual addition of registers, they automatically balance the registers based on their findings. This work lets users specify locations where they want the registers should be added. A register will be added to the location specified

and all the parallel paths are going to be balanced.

3.3 Deleting Registers

This feature lets the users delete registers after a node (if exists). This too works on the same principles as above but the implementation approach is not same. We delete register after a node and balance all parallel paths. Users can specify a location from where they wish to remove registers and all the paths after that will be taken care by the software.

3.4 Complexity

Since we implement it as a graph with adjacency list, for each addition or deletion we have to traverse the entire graph so as to locate the node and all its parallel paths. So the complexity of each add and delete is O(E+V) where E is total edges and V is total vertices. Moreover the total time complexity is dominated by the retiming algorithm which itself takes $O(V^3)$.

Chapter 4

Experiments and Results

We have divided it into two part. First one focuses on the effectiveness of the new implementation proposed in contrast with regular retiming algorithm, And the next part considers register adding an effective tool for breaking the critical path.

Here we have used an x64 CPU to compare the time taken by regular and the optimized Warshall algorithm. For each node of given type the experiment has been run three times and average is taken among all of them. The results show a huge improvement over the conventional cubic complexity distance finding algorithm with improvement in space as well. It takes only a fraction of the original algorithm and produces the desired results. The table below depicts the time taken by the regular vs the optimized algorithm.

Table 4.1 Optimized vs Regular Retiming Algorithms

Nodes	Optimized(Seconds)	Regular(Seconds)
16	0.00	0.01
124	0.32	1.03
163	0.68	2.31
172	0.74	2.70
186	0.97	3.44
199	1.10	4.20
210	1.32	5.73
232	1.91	6.60
416	8.71	38.90
430	10.10	45.01
510	21.90	132.01

The next set of results are based on critical path minimization where we try to find the critical path and try to add registers based on the observations. We try to see if the thing go our way if we add registers and try to reduce the critical path.

We have chosen two circuits with 200 and 500 nodes are tried to figure out all the critical path and added registers in all those. The below table shows the results after critical path minimisation and we implemented the results in Xilinx Vivado 2018.2. Here the latency has been varied from 10 ns to 10000 ns and each experiment has been performed 3 times and avergae values have been captured. The results were generated and then converted using an organic bench to verilog converter and then tested using Xilinx Vivado.

All the experiments are performed on intel x64 machine with 8GB primary memory and 3GHz refresh rate. Results may very depending on the architecture used.

First we gave the input clock a time period of 10ns and obtained the results below. We can see for very small clock periods the results are very insignificant.

Table 4.2 Time Taken (ns) for 10ns scale

Nodes	Time(Before)	Time(After)	Difference
200	47.15	47.16	0.01
500	48.78	49.6	0.82

Then we increase the scale and increase the clock by 10 times. We have the results as follows.

Table 4.3 Time Taken (ns) for 100ns scale

Nodes	Time(Before)	Time(After)	Difference
200	495.60	495.77	0.11
500	497.01	498.69	1.68

The results for 1000 ns are way more prominent than the results we received against 100 ns. We further take a step and increase the clock by 10 times.

Table 4.4 Time Taken (ns) for 1000ns scale

Nodes	Time(Before)	Time(After)	Difference
200	4997.60	4996.5	1.10
500	4998.68	4997.69	1.01

And to see more prominent results we increase 10 times more. Here the results were more significant.

Table 4.5 Time Taken (ns) for 10000ns scale

Nodes	Time(Before)	Time(After)	Difference
200	49997.80	49996.5	1.30
500	49999.68	49997.69	2.01

Here we see adding registers to break critical path is not much of a help. This happens since we are preforming it on small circuits. On a large scale it is certainly going to create an impact.

Chapter 5

Conclusion

This paper throws light on a way to implement standard register balancing problem using much optimised Floyd Warshall algorithm which has standard global retiming theorem in heart of it. This paper moreover shows how [1] can be used to insert and remove register and later can be balanced.

We observe that the method used here to break the critical path is theoretically strong but is not as prominent when its implemented. This is due to the size of examples taken. Taking bigger examples is definitely going to improve the slack time complexity of the circuit under observation.

References

[1] C. Karfa, "Automatic register balancing in model-based high-level synthesis," pp. 43–49, Aug 2015.

[2] C. S. Bontu and E. Illidge, "Drx mechanism for power saving in lte," *IEEE Communications Magazine*, vol. 47, no. 6, 2009.

[3] "DRX parameters in LTE," 3GPP TSG-RAN, Tech. Rep., 2007.

[4] S. Kim, D. Yun, and K. Chung, "Video quality adaptation scheme for improving qoe in http adaptive streaming," in *Information Networking (ICOIN), 2016 International Conference on*. IEEE, 2016, pp. 201–205.

[5] "Global mobile data traffic forecast update, 2015?2020," Tech. Rep., February 3, 2016.

[6] T.-Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari, "Confused, timid, and unstable: Picking a video streaming rate is hard," in *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, ser. IMC '12. New York, NY, USA: ACM, 2012, pp. 225–238. [Online]. Available: http://doi.acm.org/10.1145/2398776.2398800

[7] A. Begen, T. Akgul, and M. Baugher, "Watching video over the web: Part 1: Streaming protocols," *IEEE Internet Computing*, vol. 15, no. 2, pp. 54–63, Mar. 2011. [Online]. Available: http://dx.doi.org/10.1109/MIC.2010.155

[8] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the scalable video coding extension of the h.264/avc standard," *IEEE Trans. Cir. and Sys. for Video Technol.*, vol. 17, no. 9, pp. 1103–1120, Sep. 2007. [Online]. Available: http://dx.doi.org/10.1109/TCSVT.2007.905532

[9] S. Khan, D. Schroeder, A. El Essaili, and E. Steinbach, "Energy-efficient and qoe-driven adaptive http streaming over lte," in *Wireless Communications and Networking Conference (WCNC), 2014 IEEE*. IEEE, 2014, pp. 2354–2359.

[10] C. Liu, I. Bouazizi, and M. Gabbouj, "Rate adaptation for adaptive http streaming," in *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, ser. MMSys '11. New York, NY, USA: ACM, 2011, pp. 169–174. [Online]. Available: http://doi.acm.org/10.1145/1943552.1943575

[11] J. K. T. Thang Q. Ho and A. Pham, "Adaptive streaming of audiovisual content using mpeg dash," pp. 78–83, February 2012.