ภาคผนวก

Design Techniques for Energy Efficient Multiplier

W. Suntiamorntut
Department of Computer Engineering,
Faculty of Engineering,
Prince of Songkla University, Hatyai, Songkhla,
90112 Thailand
wannarat@coe.psu.ac.th

C. Vongchumyen
Department of Computer Engineering,
Faculty of Engineering,
King Mongkut's Institute of Technology Ladkrabang
Ladkrabang, Bangkok, 10520, Thailand
kvocharo@kmitl.ac.th

Abstract

Power consumption has become a critical concern in VLSI design, especially for portable applications. Multiplier is a fundamental operation which is executed in most clock cycle. Multiplier also has a large area, long latency and consumes a huge power. This paper presents the design technique considering at architecture, logic and circuits level such as optimization in the depth of Wallace tree addition, using modified Booth's algorithm to reduce the number of partial products, input swapping and pre-calculated signed-extension. The simulation results of the post-layout of multiplier implemented on 0.18 micron process technology show that the multiplier using our design techniques has the lowest energy consumption and also giving the best energy delay product.

Keyword: energy efficiency, multiplier, low-power design

1. Introduction

Multiplication is a basic arithmetic operation that is fundamental to digital signal processing. Whereas it consumes a huge power because the multiplier has been involved in most time slots during the execution of DSP algorithms such as the FIR filter, Discrete Fourier Transform (DFT), DCT or Linear Predictive Coding (LPC).

However, multiplier has a large area, long latency and consumes relative high power compared to other circuit components. Therefore, both low-power and low latency multiplier design have been an important part in energy efficient VLSI system design. The techniques can be applied at technology, physical, circuit and logic levels to achieve an energy efficiency. In this paper, we address at the structure, logic and circuits implementation of the multiplier. The main contribution of this work is to present the design techniques of power reduction for a parallel multiplier which has been used in CADRE-s digital signal processor proposed in [1].

The complex systems for mobile phones and portable applications are expected to support the requirements of modern features such as multimedia, high quality games and so on. The parallel or tree multiplier with a high performance carry save adder (CSA) tree has been justified for these applications[2-3]. Therefore, this research work focuses on using a coherent technique at architecture, logic and circuits level to implement a parallel tree multiplier.

The remain of this paper is organized as follows. In section2, the power awareness of CMOS circuits is discussed. The parallel multiplier architecture is described in section3 while the power saving techniques, input swapping, sign-extension and reduction in addition are presented in section4. The simulation and comparison of this multiplier has been analysed in section5. Finally, the conclusion is drawn in section6.

2. Power Awareness in CMOS

In digital CMOS circuit, there are three major sources of power dissipation as shown in the equation 1[1]:

$$P_{avg} = P_{switching} + P_{shortcircuit} + P_{leakage} \quad \ldots (1)$$
$$= \alpha_{0->1} \cdot C_L \cdot V^2_{DD} \cdot f_{clk} + I_{SC} \cdot V_{DD} + I_{leakage} \cdot V_{DD}$$

The switching or dynamic component of power consumption is the product of the load capacitance (C_L), clock frequency (f_{clk}), and activity factor, $\alpha_{0->1}$, is used to denote the average fraction of clock cycles in which a low-to-high transition occurs and the power supply V^2_{DD}. The second term is the power dissipated

due to the direct-path short circuit current, I_{SC}, which occurs during switching when both NMOS and PMOS transistor are active. The last term is the power caused by the leakage current, $I_{leakage}$, which arises from substrate injection and sub-threshold effects. The leakage current becomes a significant problem when the fabrication technology is scaled down or when there are significant periods of idle time.

Since the switching event in CMOS circuit dominates the most power, it is extremely important to reduce this source of power dissipation. Firstly, C_L could be minimized through the choice of logic style and logic topology. Secondly, the multiplier could ignore the power dissipates according to f_{clk} by using the asynchronous circuit design where it is a clock-less system[4]. Finally, reducing the supply voltage (V_{DD}) can yield more than an order of magnitude saving which is shown clearly from the equation.

In this paper, we are focusing only on parallel tree multiplier which the multiplication can be done in one clock-cycle. This leads to treat it as a combinational logic and asynchronous circuit technique is not concerned in this paper. Meanwhile, reducing a supply voltage will be ignored because there is a trade-off between performance and power when voltage has been scaling. This paper is therefore discussed only minimized load capacitance.

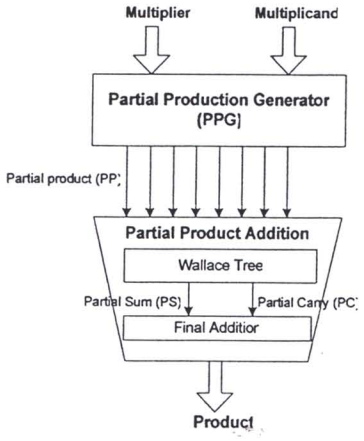


Figure 1 Parallel Multiplier Architecture

3. Parallel Multiplier Architecture

Multiplier composes of two main components: partial product generator (PPG) and the addition of those partial products (PPs) as shown in Figure 1. In order to reduce the number of PPs, modified Booth's algorithm has been used in this design, whilst the partial product addition employs the Wallace tree to produce the final partial sum (PS) and partial carry (PC). Finally, the result of multiplication is generated by the sum of PS and PC.

For a K x K multiplier, K partial products are produced and a K-input CSA tree is used to reduce them to two operands for the final addition. However, the number of partial products and logic depth of the CSA tree can be reduced when Booth's algorithm is applied. The modified Booth's algorithm[5] (radix-4 Booth's recoding) considers multiplier bits in pairs and treats the pair as a signed two's complement number. The higher radix leads to fewer partial products (PP) but requires more complex hardware and a longer time for encoding. With a radix-4 modified Booth's algorithm, the number of PPs can be reduced by half, whereas the PP generator has been built using pass-transistor multiplexer.

3.1 Partial Product Addition

In the partial product addition, several methods such as the ripple carry adder (RCA), the CSA, the CSA tree, the Wallace tree[6] and Dadda's strategy[7] are all candidates for the addition scheme in the parallel multiplier. Both Wallace's and Dadda's strategies are based on compression techniques which were first introduced by Weinberger[8]. The differences between Wallace and Dadda are that the Wallace tree tries to combine the partial product bits at the earliest opportunity, whilst Dadda's scheme combines them as late as possible and keeps the critical path (level) of the tree minimal. So Dadda's structure is simpler but has a wider CPA at the end compared to the Wallace tree. However, combining the partial product bits as soon as possible makes the Wallace tree scheme faster than Dadda.

Multi-operand addition schemes based on Wallace tree has been widely employed because of its performance. The 4-2 compressor is therefore popular in many digital multiplications. However, the compressor differs from Dadda's counter in that it is not necessary to have the pattern of M outputs drawn from 2^M inputs. An N:M compressor in essence is a variation of the Dadda counter that employs a separate path between compressor units in order to generate M final outputs using

$N>2^M$ input bits. 4-2 compressor is based on 5:3 counter using 3 bits to represent the 5 binary input bits resulting from the following equations 2-4:

$$PS = P1 \oplus P2 \oplus P3 \oplus P4 \oplus Cin \quad ...(2)$$

$$PC = Cin\,(P1 \oplus P2 \oplus P3 \oplus P4) + P3\,/(P1 \oplus P2 \oplus P3 \oplus P4) \quad ...(3)$$

$$Cout = P1(P2 \oplus P4) + P2\,/(P2 \oplus P4) \quad ...(4)$$

3.2 Final Addition

Four-bit carry-look-ahead tree unit is used to form the 40-bit carry-look-ahead tree by 10 blocks of these 4-bit units where the delay time is shorter than a conventional carry-look-ahead. The carry propagates from the bottom (C_0) bit to the top carry bit (C_4) of the 4-input unit within $3t_{mux}$ and used $12t_{mux}$ for the delay carry-chain of the 40-bit carry-look-ahead tree. Therefore the delay of a W-bit CLA-tree implemented by 4-bit unit is $((W/4)+2)t_{mux}$ as shown in Figure2 for an 8-bit adder where the multiplexer generating the sum is shown only for the LSB in each 4-bit unit.

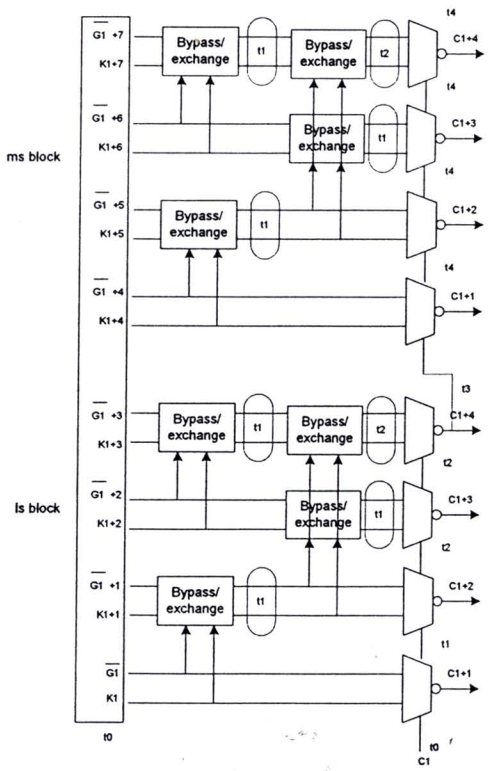


Figure 2 The structure of 4-bit CLA tree block

The adder as shown in Figure 2 has been used to form the final product by adding partial sum (PS and partial carry (PC).

4. Power Saving Techniques

4.1 Input Swapping

The switching activities of the functional blocks dominate a large portion of power consumption in a parallel multiplier where all activities inside each block depending on the data. One of two inputs data is sent to Radix-4 modified Booth's encoding block where the multiplier is decoded to generate the partial products. It can be viewed as a digit-set conversion with the 2 bits denoting the number 0 to 3 converted to the set of {-2,2}. Effectively two bits are treated as a signed two's complement number.

In the multiplication process, the control bits for the input with the smaller effective dynamic range should be used for the Booth's encoding to increase the chance of neighbouring partial products being '0'. This could result less activities.

However, this scheme has a trade-off according to an additional block as shown in Figure 3 to determine when the inputs should be swapped. It uses to detect the dynamic ranges of the input data. This block makes a decision whether the two input data paths should be exchanged or remain unchanged.

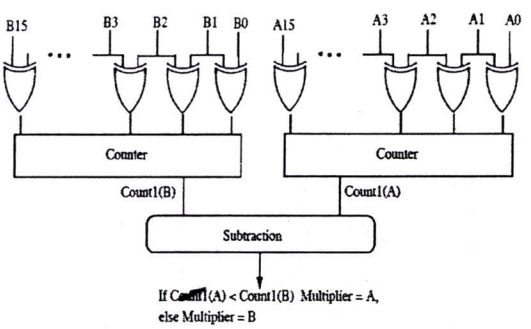


Figure 3 Additional block for input swapping

The simulation result of a 16-bit additional block shows that it takes about 1.4ns to generate the select signal for switching the inputs of the multiplier while the power dissipates at about 9mW (where the data is fetched every 1.5ns). Therefore, these performance and power consumption information of the input swapping technique has to be considered at the architectural level before adopting in the design.

4.2 Sign Extension

Most DSPs have a 16x16 multiplier located in the 40-bit datapath. So the sign-extension is

required to produce the 40-bit result. The eight PPs as shown in Figure 4 are the output of the PPG which produces eight 16 bit PPs (which is the same as the multiplicand length). However, the output is expected by a general DSP to be in 40-bit format. So each PP has to be sign-extended to 40 bits prior to their addition. If we represent logic 1-bit as a one cell, the area inside the red border shown in Figure4 is used to compute the sign-extension. The energy saving approach in the multiplier is to reduce the logic for the sign-extended bits by using a pre-calculated sign extension, as shown in Figure 5, over a group of 4 PPs. This pre-calculated number assumes that all partial products negative and thus have a sign bit of one. Thus we can save the large amount of the logic for the sign-extended calculation.

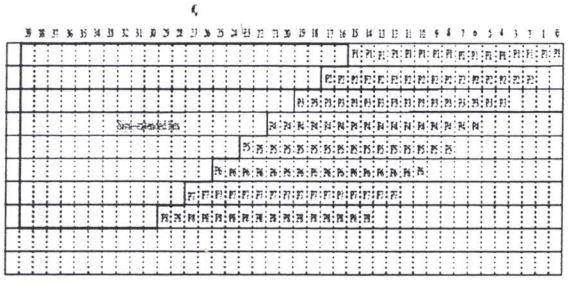


Figure 4 Structure of eight PPs addition

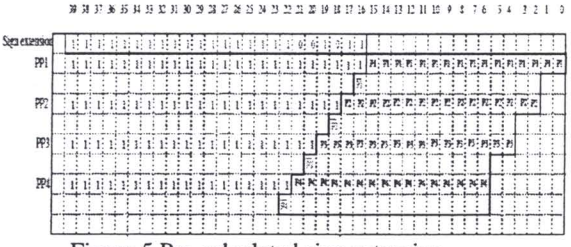
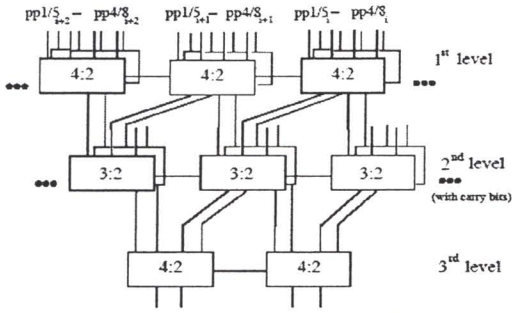


Figure 5 Pre-calculated sign extension

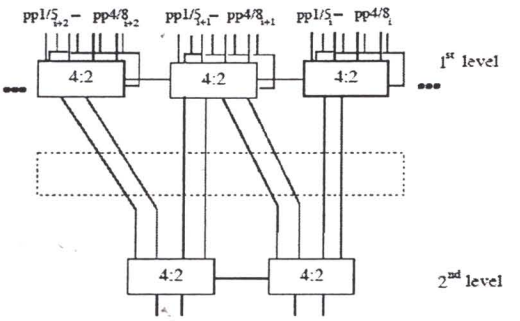
In Figure 5, the assumed sign extension for PP1 to PP3 are shown and when added yield the sign extension constant shown along the top line. If the real PP has a most significant bit (msb) equal to *one*, no adjustment is necessary. In contrast, when the real MSB is *zero,* adjustment is required equivalent to all the 'one' bits in the extension being flipped back to zero. This is achieved by adding a Px (x = 1 to 3) in the bit position shown; it should be noted that the '0's in the sign extension constant (0XFFFFCB) makes it less likely that the carry chain will extend across all bits in the extension constant. With this technique, the constant number required can be computed before the multiplication takes place in the hardware. This allows the addition hardware of the sign-bit part to be reduced.

4.3 Reduction in Wallace Tree

The Wallace tree has been applied in the addition of multiplier instead of the well-known Dadda's strategy as the partial products combining takes place as late as possible. Therefore, Wallace's method is faster than Dadda's. The addition of the PPs is usually done in 3 levels of compressors as shown in Figure 6(a). At the top level, two groups of compressors compress a group of 4 PPs to two outputs. In the second stage, two groups sum the four outputs from the top level in 3-2 compressors, with 4-2 compressors used at the final level. If the multiplier bits require a -a or -2a, then the multiplicand bits are inverted and +1 is input to the addition tree to form the two's complement input to the second stage. In this multiplier design, we propose to save the sign bits from Booth's encoding and add it up later in the final addition. Consequences, the second level compressors can be omitted as the sign bits are forwarded directly to the 4-input adder, as shown in Figure 6(b).



(a) Conventional Wallace Tree



(b) Optimal Wallace Tree

Figure 6 Wallace Tree Structure

A significant reduction in logic arises from losing the second level compressors and this

both improves performance and saves considerable circuitry as can be seen in Figure 6(b).

As a result of using this tree structure topology, the number of stages traversed by each input is approximately the same for all inputs. This leads to a balanced delay tree and results in less switching activity due to input skew.

5. Simulation Results

The design is implemented on a 0.18μm CMOS process having 6-metal layers and runs from 1.8V. The results presented here in this section are the results of simulating the layout using Nanosim eda tool and the test inputs are the random number. The multiplier can produce a PS and PC every 1.5ns in the worst case of random inputs and consumes an average power of only 10.8mW in the PPG & PP addition (excluded the input swapping and final addition). Then later, the completed multiplier (included input swapping) has been simulated to produce the final result using a data fetching rate at 200MHz. A comparison against other designs is difficult because of the differences in term of process technology and bit width.

However, selecting some other low energy designs that have been reported such as the 16x16 multiplier (32-b output) on a 0.09μm process technology running from 1.2V[9], the 4x4 wave pipeline on a 0.18μm process running from 1.8V[10], a 16x16 pass transistor multiplier (32-bit output) on a 0.8μm running from 3.3V[11] and a 16x16 multiplier (40-bit output) on a 0.13μm process running from 1.2V[12]. To get some idea of the relative merits of this multiplier, all results have been scaled to a 0.18μm geometry running from 1.8V. Whilst scaling does not take into account all effects, the results do give an indication of the energy efficiency of the different multipliers.

Table 1 shows the second lowest energy consumption (PDP) and energy delay product (EDP) of the multiplier described here (including the time to add and the adder energy) compared to others reported; it demonstrates a good compromise between performance and power for the 0.18μm process used. The power delay product can be useful for comparisons in which absolute energy values are not known whilst the energy delay product is normally used when the circuit-speed is important for an energy efficiency comparison.

Types	Scaled Power to 0.18um @1.8V (mW)	Speed (MHz)	Energy (PDP) (pJ)	Energy-Delay Product (EDP) (pJxns)
This Multiplier	26.08	200	130.4	652.0
[9]	49.50	500	99.0	198.0
[10]	74.48	167	446.8	2,681.3
[11]	11.34	44	257.0	5,855.5
[12]	201.9	435	464.4	1,068.0

Table 1 Comparison of EDP of multiplier

6. Conclusion

In this paper, the design techniques for energy efficient multiplier have been described. As a result, this multiplier can operate with energy efficiency when running the simulation using the random number as the test inputs. It shows that the design techniques of power reduction at the architectural, logic and circuits level proposed in this paper can make a good contribution.

Acknowledgement

This work has been developed and used in Configurable Asynchronous DSP for Reduced Energy – successor (CADRE-s), APT group, School of Computer, University of Manchester, UK.

7. Reference

1. W. Suntiamorntut, "Energy Efficient Functional Unit for a Parallel Asynchronous DSP", *Ph.D. Thesis, University of Manchester,* 2005.
2. A. A. Katkar and J. E. Stine, "Modified booth truncated multipliers", *Proceedings of the 14th ACM Great Lakes Symposium on VLSI*, pp. 444-447, 2004.
3. A. P. Chadrakasan and R. Broderson, "Low Power CMOS Design", *John Wiley & Sons Inc.,* ISBN 0780334299, 1997.
4. J. Sparsø and S. Furber, "Principles of Asynchronous Circuit Design - A Systems Perspective", *Kluwer Academic Publishers*, 2001.
5. B. Parhami, "Computer Arithmetic: Algorithms and Hardware Designs", *Oxford* University Press, 2002.
6. C.S. Wallace, "A Suggestion for Fast Multipliers", *IEEE Transactions Electronic Computer*, vol.EC-13, Feb., pp. 14-17, 1964.

7. L. Dadda and D. Ferrai, "Digital multipliers: A unified approach", *Alta Frequenza,* vol. 37, Nov., pp.1079-1086, 1968.
8. A. Weinberger, "4:2 Carry-Save Adder module", *IBM Technical Disclosure Bullentin*, vol.23, Jan., 1981.
9. B. R. Zeydel, V. G. Oklobdzija, S. Mathew, R. K. Krishnamurthy and S. Borkar, "A 90nm 1GHz 22mW 16x16-bit 2's Complement Multiplier for Wireless Based-band", *Symposium on VLSI Circuits Digest of Technical Papers*, pp.235-236, 2003.
10. J.B. Sulistyo and D. Sam Ha, "5GHz pipelined multiplier and MAC in 0.18um complementary static CMOS", *ISCAS'03*, May, pp.117-120, 2003.
11. C.F. Law, S.S. Rofail and K.S. Yeo, "A low power 16x16-b parallel multiplier utilizing pass transistor logic", *IEEE Journal of Solid-State Circuits*, Vol.34,
No.10, Oct., pp.1395-1399, 1999.
12. S. Agarwala et al, "A 600MHz VLIW DSP", *IEEE International Solid-State Circuits Conference Digest of Technical Papers*, vol.1, pp.56-444, 2002.

Design and Implementation of an Energy Efficient, Parallel, Asynchronous DSP

W.Suntiamorntut[1], L.E.M.Brackenbury[2] and Jim Garside[2]
[1] Department of Computer Engineering,
Prince of Songkla University
Hatyai, Songkhla, 90112, Thailand
[2] School of Computer Science, University of Manchester
Manchester, M13 9PL, UK
E-mail: wannarat@coe.psu.ac.th, {lbrackenbury, jdg}@cs.man.ac.uk

Abstract: Energy efficient computing in a DSP has become an important research issue in order to have a longer battery operating time to support the modern portable devices. The energy efficient functional unit has been designed and implemented for an in-house asynchornous DSP named, Configurable Asynchronous DSP for Reduced Energy (CADRE). CADRE-successor (CADRE-s) has been implemented as a full custom design and simulations are presented to successfully demonstrate the energy-efficiency of the FU. The results show that the FU designed can achieve an energy improvement by a factor of 5 in the multiply accumulator units and a factor of nearly 2 for the overall system compared with the original CADRE system. This demonstrates the importance that energy efficient logic, circuit and layout techniques contribute to a design.

1. Introduction

An asynchronous parallel architecture, named CADRE (Configurable Asynchronous DSP for Reduced Energy)[1] was designed in the School of Computer Science, University of Manchester and expected to use in portable applications. CADRE expanded instructions to give a flexible VLIW capability including a large register file, instruction buffer and four functional units. It has been design based on the sign magnitude number representation and asynchronous circuit design. Even though CADRE had an efficient architecture for DSP processors, it required a large amount of power as demonstrated in[1,2].

CADRE was implemented by exploiting four-way parallelism, as this appeared to be optimal for power reduction[3]. This was based on the premise area can be traded for increased speed because silicon area is rapidly becoming less expensive. As well-known, arithmetic operations, such as multiply, add and multiply-accumulate are frequently performed and consumed hungry power. From the previous work using random plus speed data, a large percentage of power was found to be dissipated in the Multiply Accumulator Units (MAC units) amounting to about 50% of the overall power consumption.

Therefore, the functional unit (FU) has been re-designed and re-implemented to demonstrate the energy saving improvement possible. To reduce the power consumption in the new FU, the major dominant components, the multiplier and adder, are designed and implemented with coherent low power techniques. The new four-way parallel asynchronous DSP which has been designed, implemented and tested is called CADRE-s (CADRE successor) and will be referred to as CADRE-s throughout in this paper.

This paper is organized as follows. In Section II we describe the CADRE-s Top-Level Architecture. Coherent low power techniques using in the new FU is illustrated in Section III. The implementation is shown in Section IV. Finally, Section V concludes this paper.

2. CADRE-s Top-level Architecture

The top-level architecture for CADRE-s has been designed to demonstrate the energy efficiency of the functional unit and the other advantageous features of CADRE-s such as four-way parallelism and configurable memories. In this top-level architecture, the 32x64 bit configuration memory has been attached to each FU and it stores the opcode. In contrast with the original CADRE, the operands of each FU in the current architecture are fetched directly from on-chip RAMs (OPA and OPB). The new system consists of four FUs connected together with a global bus named the Global Interface Functional Unit (GIFU), whilst each pair of FUs are connected locally via the bus named Local Interface Functional Unit (LIFU). The output data from each FU is stored in another on-chip RAM. Two RAMs are used for the top level control. One is the top-level 14-bit instruction and is stored in a program memory. The second one is the address RAM which effectively performs the function of a 16-bit program counter (PC).

CADRE-s employs a scan path structure for downloading instructions/data and uploading results as shown in Figure1. There are five separate serial scan paths in the design, one is for the address and program memories and the others used for the operand, configuration and result memories. This makes the system fully testable as internal states can be controlled and monitored.

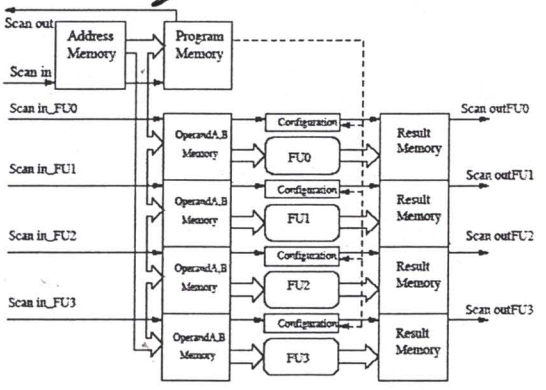


Figure1 CADRE-s Data Organization

2. 1 MIMD with VLIW encoding

In CADRE-s, each FU has its own operands, configurable and result memories. Hence, each FU can

execute independently. It differs from the FU that has been used in an asynchronous superscalar[4]. It has shared memories and registers. A special mechanism for the instruction dispatch unit is therefore required. Because CADRE-s contains four FUs which work independently, the instruction is relatively long. To minimize the length, the instruction has been divided into two parts. One part is stored in the configuration memory and the other part in the top level instruction memory. The configuration memory contains the FU opcode, input/output selection, and the shift condition whilst the top level instruction contains four enable signals plus four accumulator write enable control signals and a common 6-bit address for the configuration memory. Storing the 6-bit address of configuration memory not only reduces the number of instruction bits but the instruction can be compressed. This means that the instructions can be recalled when the same instructions but different data are required. Most DSP algorithms can take advantage of this feature to reduce the size of the program.

2. 2 Five Stage Asynchronous Pipeline

The CADRE-s pipelined processing unit is organized using self-timing techniques. An asynchronous pipeline operates at a variable rate determined by current conditions, unlike a single clock system, where the whole pipeline will be clocked at a rate determined by the worst-case delay in the slowest stage. Although, a multi-clocked system has been proposed in [5-6] to allow a variable rate operation, this method is limited by the number of the various clock cycles which are generated by the clock generator. In a clockless system, the next instruction can start as soon as the previous result is generated. Therefore, the self-timed system is able to operate at the average-case performance rather than worst-case.

3. Low Power Design Techniques

Two novel techniques have been employed in the implementation. The first is to build the functional units (FU) using pass transmission gate (PTG) logic; such logic promises significantly lower power than conventional CMOS whilst delivering high performance. The second unusual technique employed it that the whole system is clock-free. Clockless (or asynchronous) logic is used to eliminate clock generation, buffering and distribution – a major power user – at the system level. Each functional unit is responsible for its own timing; data is passed in and out using handshake signals. This means that a functional unit which is not in use dissipates almost no power. It also allows the implementation of 'unusual' DSP functions – such as Hamming distance calculation or signal clipping – in an energy efficient manner. If evaluation in one unit is slow the whole system will adapt on a cycle-by-cycle basis. The final advantage of asynchronous logic here is that it adapts automatically as the supply voltage is changed; a reduced input voltage gives slower processing but much greater energy efficiency.

4. Implementation

CADRE-s is implemented on a 0.18µm CMOS process having 6-metal layers and runs from 1.8V. The tests described seek to demonstrate the energy efficiency of the FU designed by the coherent low energy design techniques described in the previous section. The kernel benchmarks are translated into binary codes by the CADRE assembler. A Verilog module then rearranges this binary code into the format for serially shifting into CADRE-s. The CADRE-s die, shown in Figure2, measures 4.5 x 3.9mm and contains over 230,000 transistors plus 14 RAM memories of 2k x 16 bits plus a further 4 RAM memories of 64 x 32 bits. The results show that the FU designed can achieve an energy improvement by a factor of 5 in the multiply accumulator units and a factor of nearly 2 for the overall system compared with the original CADRE system.

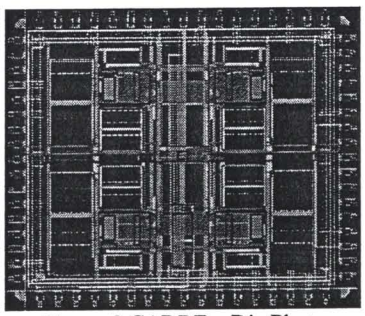


Figure 2 CADRE-s Die Photo

5. Conclusion

CARE-successor (CADRE-s) has been implemented as a full custom design and simulations are presented to successfully demonstrate the energy-efficiency of the FU. This demonstrates the importance that energy efficient logic, circuit and layout techniques contribute to a design.

6. Acknowledgement

This work was funded by EPSRC grant GR/S61270/01 and the authors are grateful for this support. The authors are also grateful to Dave Clark and Jeff Pepper for encouragement in this work.

References

[1] M.Lewis and L. Brackenbury, "CADRE: An Asynchronous Embedded DSP for Mobile Phone Applications", *Design Automation for Embedded Systems*, Vol.6, No.4, pp.451-475, 2002.

[2] M. J. G. Lewis, "Low Power Asynchronous Digital Signal Processing", *Doctor of Philosophy Thesis*, Faculty of Science and Engineering, University of Manchester, 2000.

[3] Anantha P. Chandrakasan, Robert W. Brodersen, "Minimizing Power Consumption in CMOS Circuits", Proceedings of the IEEE, vol.83, Apr., pp.498-523, 1995.

[4] D.K.Arvind and Robert D. Mullins, "A Fully Asynchronous Superscalar Architecture", *International Conference on Parallel Architecture and Compilation Techniques*, Oct., p.17-22, 1999.

[5] D. Peiliang, Y. Rilong, X. Hongbo and Y. Chengfang, "Multi-clock driven system: a novel VLSI architecture", *Proceedings 4th International Conference on ASIC*, Oct., pp.555-558, 2001.

[6] M. Singh and M. Theobald, "Generalized latency-insensitive systems for single-clock and multi-clock architectures", *Proceedings on Design, Automation and Test in Europe Conference and Exhibition*, vol.2, Feb., pp.1008-1013, 2004.

Survey of finding empty space algorithm for partial reconfigurable FPGAs

Sasithorn Somvathee
Dept. of computer engineering, faculty of engineering
Prince of Songkla University
Hatyai Songkhla Thailand 90112
ssasatorn@hotmail.com

W. Suntiamorntut
Dept. of computer engineering, faculty of engineering
Prince of Songkla University
Hatyai Songkhla Thailand 90112
wannarat@coe.psu.ac.th

Abstract—**Partial reconfigurable FPGAs can dynamically modified their logic and interconnect configuration at runtime without affecting each other. Finding the available empty space for incoming tasks at runtime is the most time consuming process in the online placement algorithm. The primary goal of such an algorithm is speed and memory usage. This paper presents the survey of three finding free resources algorithms by presenting their data structure and method. A comparison of time complexity for each algorithm is also presented here.**

Keywords: online placement, partially reconfigurable FPGAs, hardware multitasking

I. INTRODUCTION

The flexibility of reprogrammability in FPGA (Field Programmable Gate Arrays) is a great advantage of this device. We can make the use of their reprogrammability in one of two ways: Compile-Time Reconfiguration (CTR) or Run-Time Reconfiguration (RTR). The FPGA configuration can be changed during the operation execution when the RTR system is applied. The configuration can be fully or partially reprogrammed.

A FPGA consists of a rectangular grid of Configurable Logic Blocks (CLBs) and interconnection between the cells. FPGA allows multiple tasks to be executed in parallel by hardware. For such systems an application is divided into smaller tasks, called hardware tasks. When the system requests its execution each of them is placed on the FPGA. After a task finishes execution, the system deletes it and its area can be reclaimed and reused by other tasks.

The task placement may be subdivided into two categories: offline and online. In the offline placement, the configuration order and location of tasks are known when the application is compiled.

On the other hand, in the online placement, all tasks are known at runtime and can be placed anywhere on the chip. The configuration of hardware tasks on the FPGA must be done on the fly. So the system searches available resources for a new task on a case-by-case basis, which is time-consuming.

Because requested tasks are known only at runtime, the system's management must face with two problems: where to place a new arrival task for execution [1], [2], [3] and how to provide communication among the modules running on the FPGA [4], [5].

This paper presents three approaches to solving the problem of finding the free space for arrival tasks on the FPGAs. The first is Staircase algorithm [1], which works by first finding all the maximal staircases and the extracting the maximum empty rectangles from them. The second, enhanced Scan Line algorithm [2] uses 2D FPGA surface model with encoding information. MKE points are defined to utilize the scanning process while looking for the maximum free rectangles. The third, Flow Scan algorithm [3] scans the maximum empty rectangles from the task edges.

In section 2, we describe the detail of these three algorithms. Then, we show the comparison of time complexity in section 3. Finally, we conclude this paper in section 4.

II. EXISTING ALGORITHMS FOR FINDING EMPTY SPACE

The free FPGA space is usually recorded as a set of rectangles because most of the hardware tasks can be fitted in a rectangular shape. There are two types of rectangles: the non-overlapping rectangles and the maximum rectangles. In the non-overlapping, algorithm has a shorter execution time but more task rejection rate than the maximum rectangles.

In this section we describe the algorithm for finding all maximum empty rectangles by using the same sample as shown in figure 1. As an example, FPGA consists of 10x10 cells with two placed tasks running. Maximum Empty Rectangle (MER) is denoted by the tuple (x, y, w, h), where (x, y) is the coordinates of its lower left corner, and (w, h) is its width and height. There are 6 maximum empty rectangles in total: (1, 1, 10, 1), (1, 1, 1, 10), (5, 1, 6, 4), (5, 1, 1, 10), (1, 7, 5, 4), (1, 9, 10, 2).

A. *Staircase algorithm*

This algorithm uses a 2D array with X number of columns and Y number of rows. This array is called area matrix. Each cell in the array represents a CLB in the FPGA. Bottom left cell is addressed (1, 1) and top right cell is addressed (X, Y).

If a CLB is used by a task, the cell is called an occupied cell, which is represented by a negative number. Otherwise it is called an empty cell, which is represented by a positive

number. Figure 2 shows the area matrix of a FPGA with two tasks placed on it. A positive number gives number of contiguous empty cells above and including that cell in that column and a negative number gives the remaining width of the task.

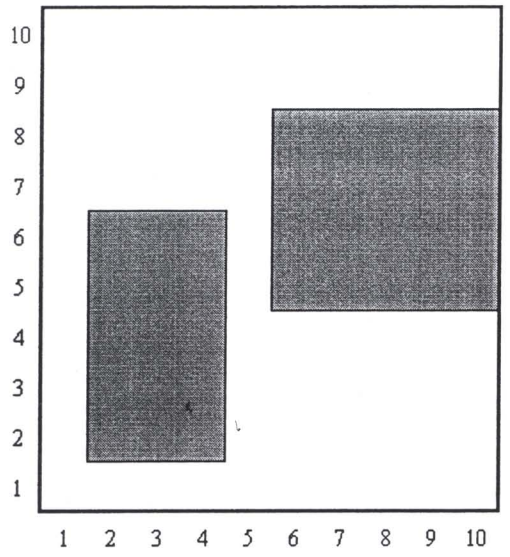


Figure 1. Example FPGA configuration with two placed tasks.

This algorithm scans the area matrix on a row-by-row basis from top to bottom and make staircase at the empty locations. We check only the maximal staircases for extracting maximum empty rectangles. A staircase is maximal if it cannot be extended down or to its right. Only the top horizontal boundary of already placed tasks and the bottom-most row will be scanned as shown in figure 2. In each scanned row, scanning is done from the left side to the right side. If the left boundary of a task is encountered, the occupied cells can be skipped. This gives considerable savings in runtime.

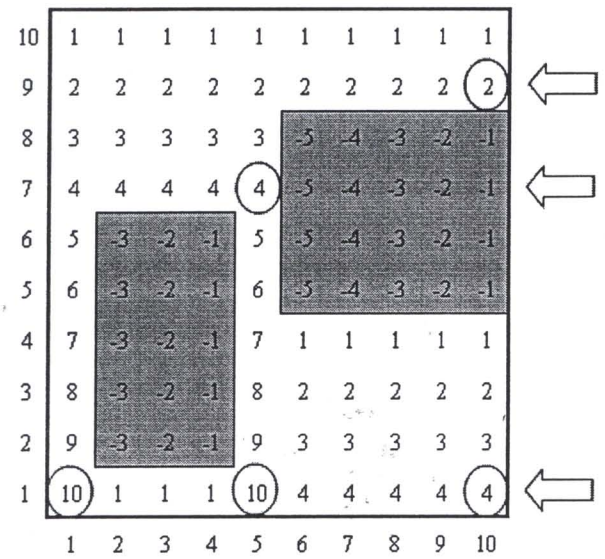


Figure 2. Modeling FPGA area matrix, the origin point of maximal staircase (circles) and the scanning row (block arrows)

In order to construct a staircase, there is only one stair in the staircase at the first leftmost positive entry of each scanned row. The height of that stair is given by positive integer stored at that location. A staircase at point (x+1, y) can be easily constructed from a staircase at point (x, y). Let (x, y') be the top right most point on the stair containing column x and (x+1, y'') be the coordinate of topmost empty cell on the column x+1. There are three possible cases to construct a staircase:

y" > y' A new stair is added to staircase and y" becomes top-left corner of the new stair

y" = y' Width of the rightmost stair is extended by one.

y" < y' All the stairs with height greater than that y" are deleted and the last one is replaced with new highest stair, which has height equal to y".

B. Enhanced Scan Line Algorithm

This algorithm also use 2D array but add one extra column to the right edge of the FPGA array, and assign value 0 to cells on that column. The occupied cell is represented by value 0 and the empty cell is represented by a positive number. A positive number in a cell gives number of contiguous empty cells at the left hand side and including that cell in that row.

Before scanning the free space with this algorithm, we have to find the Key Element, which is an empty cell with an occupied cell as its right hand neighbor, or an empty cell on the right edge of FPGA area. Next, we select the Scan Line. The Scan Line contains one or more Key Elements. Then we find the Valley Point to create a segment and choose the largest Key Element as the MKE.

The algorithm chooses the maximum value of MKEs in each column. Next, the scanning moves the tops and bottoms of multiple MKEs simultaneously. If they overlap with each other, then the subsequent steps of scanning these MKEs will be identical, and we only need to continue the scanning process for one of the MKEs to avoid any redundant scanning process.

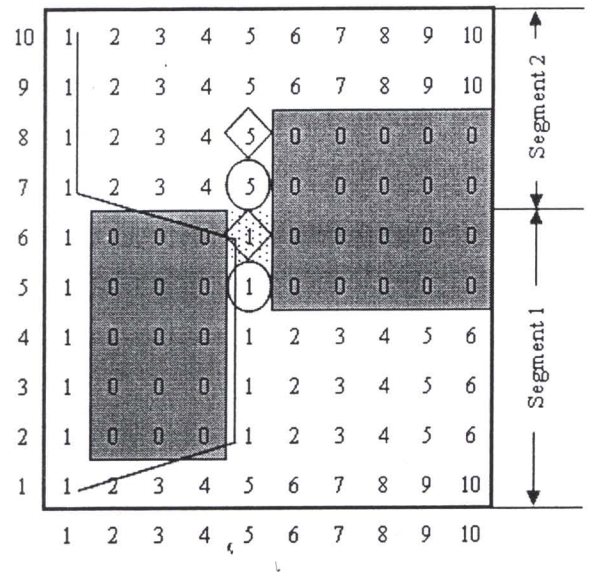
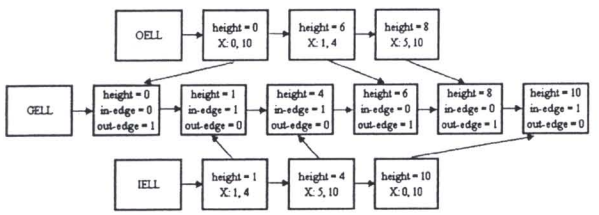


Figure 3. Modeling FPGA area matrix, MKEs (circles), Key Elements that are not maximal (diamond) and Valley Point (dotted cell) on column 5.

C. *Flow Scan Algorithm*

This algorithm uses linked list structure to store runtime information about the free space as in figure XXX. There are 4 different linked lists: general edge linked list (GELL), in-edge and out-edge linked lists (IELL and OELL), and rectangular well linked list (RWLL).



Because the linked lists is sorted when inserting or deleting task, the algorithm scans from the lowest height to the highest height of the GELL, which is the edge of FPGA. When the scanning flow reaches an out-edge, the out-edge processing is called. Only one new FRW is created. Its bottom has the same height as the out-edge.

When the scanning flow reaches an in-edge, the in-edge processing happens. The search for overlapped FRWs will start. If there is an overlapped FRW with an in-edge in the X direction, a maximum free rectangle is created by adding the height of the in-edge as a top line of the FRW and at most two new RWs can be created in case of the left side of FRW < the left side of in-edge or the right side of FRW > the right side of in-edge. No FRW will be generated if the height of the in-edge is the same as the top of FPGA.

III. COMPARISON OF TIME COMPLEXITY

The following sections provide comparisons of time of the previously described scanning algorithms.

An FPGA has x number of columns and y number of rows.

In the staircase algorithm, let n be the number of rows in FPGA which lie immediately above top boundary of some tasks. A staircase needs to be constructed at every column in a row. So, time complexity is O(xn). The worst case performance of staircase algorithm is O(xy).

For the scan line algorithm, it takes time O(y) to find all the MKEs of a Scan Line. If there are n scan lines, then the time complexity is O(ny). The worst case time complexity of scan line algorithm is O(xy), which is the same as the staircase algorithm.

On the other hand, only the task edges are processed. Let n be the number of placed task. The time complexity is O(n). The worst case for the flow scan algorithm is when all edges of n placed tasks are located on different height. This makes the worst case complexity of flow scan algorithm is O(y).

IV. CONCLUSION

In this article we provide descriptions of three algorithms for finding the maximum free rectangles on the partially reconfigurable FPGAs. We have presented a comparison of theses algorithms, highlighting their features and differences. Lastly since the use of online placement is important in the reconfigurable system, there are still many challenges that need to be met.

REFERENCES

[1] M. Handa, and R. Vemuri, "An efficient algorithm for finding empty space for online FPGA placement". In *Proc. of the 41st annual conference on design automation*, San Diego, CA, June 2004, pp 960-965.

[2] J. Cui, Q. Deng, X. He, and Z. Gu, "An efficient algorithm for online management of 2D area of partially reconfigurable FPGAs". In *Proc. of the conference on Design, automation and test in Europe*, Nice, France, April 2007, pp 4-6.

[3] Y. Lu, T. Marconi, G. Gaydadjiev, and K. Bertels, "An efficient algorithm for free resources management on the FPGA". In *Proc. of the conference on Design, automation and test in Europe*, Munich, Germany, March 2008, pp 1095-1098.

[4] C. Bobda, A. Ahmadinia, "Dynamic Interconnection of Reconfigurable Modules on Reconfigurable Devices". In *IEEE Design and Test of Computers*, vol. 22, no. 5, pp. 443-451, September/October, 2005.

[5] M. Tomono, M. Nakanishi, S. Yamashita, N. Nakajima, and K. Watanabe. "A new approach to online FPGA placement". In *40th annual conference on Information Sciences and Systems*, Princeton, USA, March 2006, pp 145--150.

Performance analysis of an efficient algorithm for free resources management on the FPGA

Sasathorn Somvathee
Department of computer engineering
Prince of Songkla University
Hatyai, Songkhla 90112 Thailand
ssasatorn@hotmail.com

W. Suntiamorntut
Department of computer engineering
Prince of Songkla University
Hatyai, Songkhla 90112 Thailand
wannarat@coe.psu.ac.th

Abstract— **The most time consuming of an online task placement is to find the available free space on the FPGAs. The flow Scan algorithm is recently purposed for finding a complete set of maximum empty rectangles. The performance of this algorithm compared with others is reported in this paper.**

Keywords: online placement, partially reconfigurable FPGAs, hardware multitasking

I. INTRODUCTION

Partially reconfigurable FPGA (Field Programmable Gate Array) allows their logic and interconnection between the cells modified at runtime without affecting each other. For such system an application is divided into smaller tasks called hardware tasks. Hardware tasks run concurrently on the FPGA. When a task finishes execution, the system removes it from the FPGA and its area can be relocated and reused by incoming tasks. Online partial reconfiguration allows the sequence of tasks to be performed unpredictably because the FPGA controller can make a decision online. Partially reconfigurable FPGA can also support the concept of dynamically dataflow reconfigurable coprocessor architecture.

In the offline placement, the sequence of request tasks is known in advance. In contrast with this method, the flow of tasks is known at runtime. We call this environment *online placement.* Various optimization algorithms have been applied to obtain good quality placements. The system needs to handle each task on a case-by-case basis. The placement time is an overhead on total execution time of the application. Finding and maintaining empty space on the FPGA is the most time consuming part of a placement algorithm. Thus a fast algorithm is required to efficiently manage free space for fast task placement.

There are three approaches for maintaining the empty space on the FPGA device: as a list of non-overlapping rectangles, a list of maximum empty rectangles, or a list of vertices, each with its pros and cons. Management a list of maximum empty rectangles is to fit more tasks on a given area than a list of non-overlapping rectangles. But it is often time consuming to maintain a complete set of maximum empty rectangles. In this paper, we implement an efficient algorithm for finding the complete set of maximum empty rectangles called Flow Scan algorithm [1].

The literature review of free space management in FPGA is explained in section 2. This paper details the flow scan algorithm in section 3. Then we discuss the problems found in the implement phase in section 4. In section 5, we present the simulation result and compare with the algorithm reported in [1]. Finally, we conclude this paper in section 6.

II. RELATED WORK

In the dynamic reconfigurable system, low area utilization [6,7] is the result of resource fragmentation in FPGA. An efficient algorithm to find empty space on the FPGA could help the defragment and task relocation. In this research area, we define MER as *Maximal Empty Rectangle* that means the empty rectangle that cannot be covered fully by any other empty rectangle.

Finding empty space is the basic fundamental problem in computational geometry. Jin Cui, et. al. in [8] proposed an efficient algorithm for finding the complete set of MERs in FPGA. Their works gave a better result compared to Scan Line Algorithm (SLA). However, in [9] showed that maintaining empty space as maximal rectangles leads to better utilization of resources. Therefore, Handa, et. al. in [1] reported a good quality results of their algorithms using maintaining maximal rectangles.

III. FLOW SCAN ALGORITHM

A. *Definitions[3]*

The lower Y coordinate of each placed task and top of FPGA are defined as *in-edge* while the *out-edge* is used for the higher Y coordinate and the bottom of the task. The direction of scan flow is from in-edge to out-edge.

Rectangular well (RW) is defined as temporally rectangles without top lines during the scanning process. *Formed rectangular well (FRW)* is any RW that can only be expanded upwards. Only FRW is recorded and temporal RW will be removed when there are several RWs with the same X coordinate crated during the scanning process. *Maximum free rectangular* is defined as rectangle that top, bottom, left and right edge cannot be expanded.

B. Data Structure

The flow scan algorithm uses linked list to store the runtime information. Figure 2 shows the linked lists, representing the situation as depicted in figure 1.

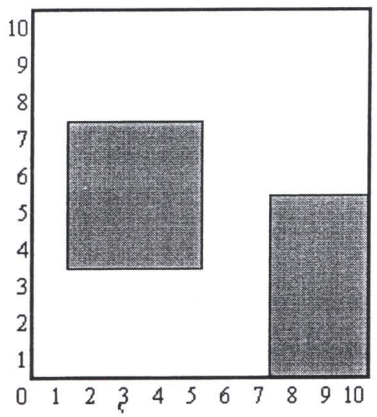


Figure 1. FPGA surface with two placed tasks.

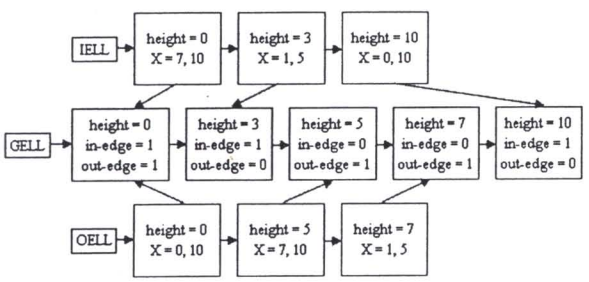


Figure 2. Linked lists.

For each placed task, its lower Y coordinate is defined as in-edge and its higher Y coordinate is defined as out-edge. The bottom and top of FPGA are defined as out-edge and in-edge by default respectively.

A rectangular well (RW) corresponds to the temporary rectangle without a top line, which created during the scanning process. For any RW that can be only expanded upwards is defined as formed rectangular well (FRW).

A maximum empty rectangle is a rectangle whose top, bottom, left and right cannot be expanded. In this paper, it is abbreviated as (left, right, bottom, top).

There are two scanning process in the flow scan algorithm: in-edge processing and out-edge processing.

The out-edge processing happens when the scanning flow leaves an out-edge. Only one new FRW is created. Its bottom has the same height of the out-edge.

When reaching an in-edge, the in-edge processing is called. The search for overlapped FRWs will start. If a FRW is overlapped with an in-edge in the X direction, a maximum empty rectangle is created by adding the height of the in-edge as the top line of new FRW. And, then, if the height of the in-edge is not the full height of FPGA, at most two new RWs can be created for the non-overlapping area within the FRW.

IV. Problems Found While Implementing

In this section, we discuss about the problems, which found at the implement phase.

The first problem is to find overlapped FRWs with an in-edge. This problem solved by comparing the X coordinates between FRW and in-edge. The overlapped FRW occurs if the right side of FRW is greater than the left side of in-edge or the left side of FRW is less than the right side of in-edge. Figure 3 shows the overlapped and non-overlapped FRWs.

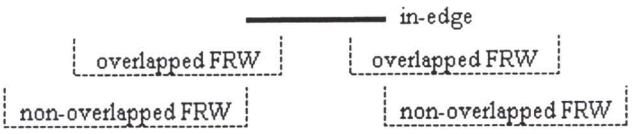


Figure 3. Example of overlapped and non-overlapped FRWs

Next problem is how to find the bottom of new FRW. In the out-edge processing, we can easily use the height of the edge as the bottom of new FRW. Meanwhile in the in-edge processing, new FRW is only created when it has overlapped FRWs. So we use the bottom of overlapped FRWs (dot cells) as the bottom of new FRW as shown in figure 4 (b).

Similar to the previous problem, what is the X coordinates of new FRW? In the in-edge processing, when it has overlapped FRWs and the non-overlapped part is on the left hand side, we use the left of overlapped FRW as the left of new FRW and the left of in-edge as the right of new FRW. If the non-overlapped part is on the left hand side, we use the right of in-edge as the left of new FRW and the left of overlapped FRW as the right of new FRW.

On the other hand, in the out-edge processing, the left and right of new FRW come from the leftmost of current FRW which the right of current FRW is equal to the left of out-edge and the rightmost of current FRW which the left of current FRW is equal to the right of out-edge as depicted in figure 4 (a).

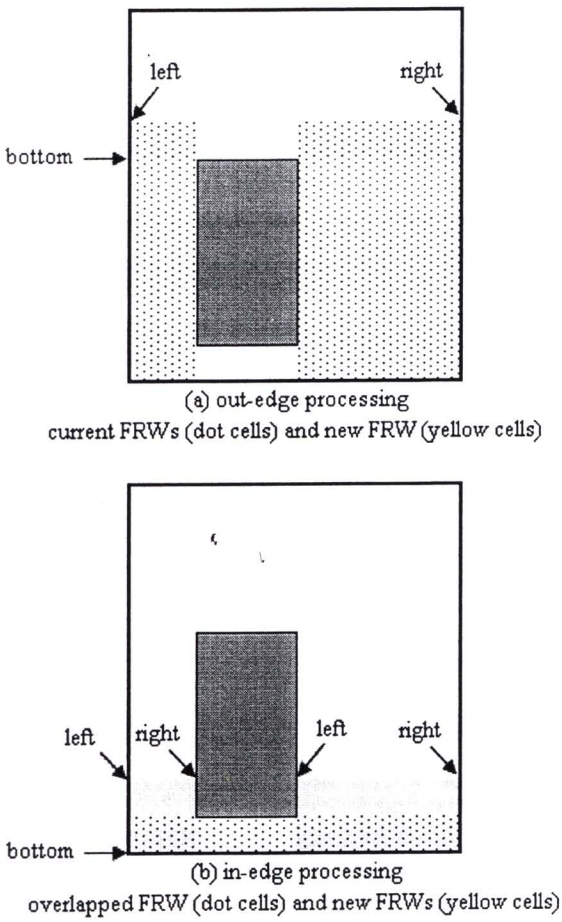


Figure 4. FPGA surface with placed task.

Finally, when the FRW is created, the important thing is to verify that the new one is not duplicated or overlapped with the current in lists. Also, when the maximum empty rectangle is recorded in the in-edge processing, we have to check that the bottom and the top is not the same height.

V. SIMULATION RESULT

The algorithm was implemented in C, and evaluated under Ubuntu 8.04 running on Intel Core 2 Duo 2.0 CPU 1.8 GHz with 2 GB main memory. We integrated with the simple online placement algorithm. The placement algorithm uses first fit policy to find the free space for incoming tasks from a complete set of maximum empty rectangles generated by the scanning process.

In order to compare with the original paper (1), we use the same simulation test. We start from 100x100 configurable logic units of FPGA. 1000 tasks were generated randomly by using the output of the previous scanning. Due to the simulation is not considered on the task rejection rate. One of the maximum empty rectangles is selected randomly and the size of the new task is randomly generated within the selected maximum empty rectangle. The arrival time is assigned between [5..25] time units. The task life time has 3 ranges: T_{250}, T_{500} and T_{1000}. For T_{250} the task life time is randomly chosen from the time interval [5..250]. For T_{500} the [251..500] is used and for T_{1000} the [501..1000] is used.

Because the CPU clock speed is too fast, then we use gprof program to calculate the amount of time spent in each routine. First, we compiled the algorithm with 'gcc -pg <source.c> -o <program>'. Next we ran the simulation and see the execution profile by using 'gprof <program>'. Figure 5 shows the result of gprof program.

```
File  Edit  View  Terminal  Tabs  Help
dekmaidee@dekmaidee-laptop:/media/disk/Code$ gprof fs_algo -b
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
 50.00      0.01     0.01    16959     0.59     0.59  get_overlap_list
 50.00      0.02     0.01    15341     0.65     0.97  remove_task
  0.00      0.02     0.00    41715     0.00     0.00  create_frw
  0.00      0.02     0.00    24648     0.00     0.00  record_maximum_rectangle
  0.00      0.02     0.00    15960     0.00     0.00  out_edge_processing
  0.00      0.02     0.00     7376     0.00     1.36  in_edge_processing
  0.00      0.02     0.00     5000     0.00     0.00  random_int
  0.00      0.02     0.00     2002     0.00     0.00  insert_edge
  0.00      0.02     0.00     2002     0.00     0.00  insert_gell
  0.00      0.02     0.00     1982     0.00     0.00  remove_edge
  0.00      0.02     0.00     1967     0.00     5.08  fs_algorithm
  0.00      0.02     0.00     1000     0.00     0.00  create_task
  0.00      0.02     0.00     1000     0.00     0.00  find_position
  0.00      0.02     0.00     1000     0.00     5.08  insert_task
  0.00      0.02     0.00        1     0.00     0.00  initial
```

Figure 5. Screen shot of gprof program

The algorithm is executed every time when a new task arrives or one is removed. In our simulation with 10000 tasks, the algorithm is invoked approximately 19000 times. As shown in the figure 6, the average number of microseconds spent in this algorithm per call is presented.

In the flow scan algorithm, only the task edges are processed. The worst case is when all edges of n placed tasks located on different heights. The worst case time complexity of flow scan algorithm is O(2n).

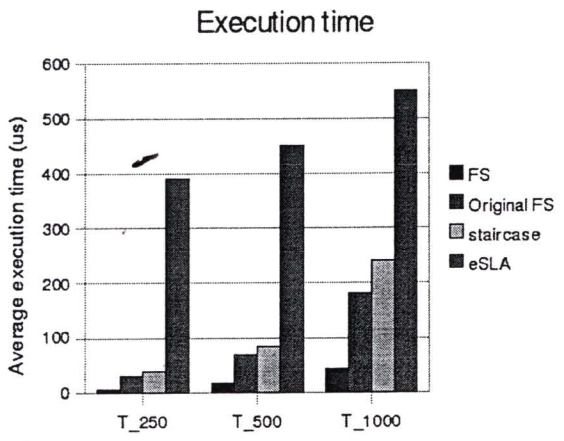


Figure 6. Execution time

VI. CONCLUSION

In this paper, we describe the scanning algorithm called flow scan algorithm. The problems and their solution are presented. Lastly we simulate the test to evaluate performance compare with the original paper.

REFERENCES

[1] M. Handa, and R. Vemuri, "An efficient algorithm for finding empty space for online FPGA placement". In *Proc. of the 41st annual conference on design automation*, San Diego, CA, June 2004, pp 960-965.

[2] J. Cui, Q. Deng, X. He, and Z. Gu, "An efficient algorithm for online management of 2D area of partially reconfigurable FPGAs". In *Proc. of the conference on Design, automation and test in Europe*, Nice, France, April 2007, pp 1-6.

[3] Y. Lu, T. Marconi, G. Gaydadjiev, and K. Bertels, "An efficient algorithm for free resources management on the FPGA". In *Proc. of the conference on Design, automation and test in Europe*, Munich, Germany, March 2008, pp 1095-1098.

[4] C. Bobda, A. Ahmadinia, "Dynamic Interconnection of Reconfigurable Modules on Reconfigurable Devices". In *IEEE Design and Test of Computers*, vol. 22, no. 5, pp. 443-451, September/October, 2005.

[5] M. Tomono, M. Nakanishi, S. Yamashita, N. Nakajima, and K. Watanabe. "A new approach to online FPGA placement". In *40th annual conference on Information Sciences and Systems*, Princeton, USA, March 2006, pp 145--150.

[6] M. Gericota, G. Alves, M. Silva and J. Ferreira, "Runtime management of logic resources on reconfigurable systems", *In Proceedings of Design, Automation and Test in Europe*, March, 2003.

[7] M. Handa, and R. Vemuri, "Area fragmentation in reconfigurable operating systems", *In Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms*, CSREA Press, June, 2004.

[8] Jin Cui, Qingxu Deng, Xiuqiang He and Zonghua Gu, "An efficient algorithm for online management of 2D area of partially reconfigurable FPGAs", *Design, Automation & Test in Europe Conference & Exhibition (DATE'07)*, April, 2007, pp.1-6.

[9] K. Bazargan, R. Kastner, and M. Sarrafzadeh. "Fast template placement for reconfigurable computing systems",