System-level Scheduling on Instruction Cell Based Reconfigurable Systems

Ying Yi[1], Ioannis Nousias[1], Mark Milward[1], Sami Khawam[1], Tughrul Arslan[1, 2], Iain Lindsay[1]

School of Engineering and Electronics [1]
University of Edinburgh, Edinburgh, EH9 3JL
Tel: (+44)131 650 5619 Email: M.Yi@ed.ac.uk

Institute for System Level Integration [2]
Alba Centre, Livingston, Scotland, EH54 7EG, U.K
Tel: (+44) 131 6505592 Email: T.Arslan@ed.ac.uk

Abstract

This paper presents a new operation chaining reconfigurable scheduling algorithm (CRS) based on list scheduling that maximizes instruction level parallelism available in distributed high performance instruction cell based reconfigurable systems. Unlike other typical scheduling methods, it considers the placement and routing effect, register assignment and advanced operation chaining compilation technique to generate higher performance scheduled code. The effectiveness of this approach is demonstrated here using a recently developed industrial distributed reconfigurable instruction cell based architecture [11]. The results show that schedules using this approach achieve equivalent throughput to VLIW architectures but at much lower power consumption.

1. Introduction

Leading-edge DSP applications such as mobile video, audio and telecom require high performance on a small energy budget. Three main existing methods to implement algorithm solutions on silicon are programmable DSP (DSP), Application Specific Integrated Circuit (ASIC) and Field Programmable Gate Array (FPGA). DSPs are highly flexible in terms of programmability. But have limited throughput, even though very long instruction word (VLIW) DSPs offers some parallelism in form of independence instruction level parallelism. Hardwired ASIC implementations consume less energy and realise the algorithm with increased parallelism but this lacks post-fabrication flexibility. Field Programmable gate arrays give a design close to the performance benefits of an ASIC solution and provide a reduction in NRE cost due to post-fabrication flexibility. However, the large number of the transistors in FPGAs used for configuration and interconnection has a huge impact on energy consumption and silicon area[1]. Reconfigurable computing is an emerging technology that combines the flexibility of FPGAs with the programmability found in General Purpose Processor (GPP)/DSPs in a unified and easy programming environment [2-10]. One such architecture, that meets the above criteria, is the one recently introduced by our research group [11].

A challenging issue in reconfigurable computing systems is the development of a suitable programming interface. In order to seamlessly integrate such architecture to existing design methodologies, a high-level programming interface is required, which will also simplify and reduce the design time (Time to Market). The high level C language is used for describing the entire application in our compiler environment. However, traditional software is sequential in nature. This poses a challenging task of taking an otherwise sequential code to extract and exploit the available parallelism. The design flow to automate the implementation of algorithms from a high abstraction level to run on reconfigurable computing system requires an efficient scheduling algorithm that can translate general purpose software code onto the reconfigurable device thereby maximising the resource utilisation.

Operation chaining merges two operations one after another in extended clock cycle to reduce the register requirement for the application. This paper proposes an efficient operation chaining reconfigurable scheduling (CRS) algorithm. The CRS algorithm is based on the list scheduling algorithm and adopts the advanced operation chaining technique and considers the effects of register assignment, power consumption, placement and routing delay against the resource and time constraints. The CRS algorithm allows high program throughput and low power consumption by ensuring that the number of dependent and independent instruction cells (ICs) is maximised at each scheduled step and at the same time the total number of clock cycles of the longest-path delay is minimised.

This paper is organised as follows. Section 2 of this paper covers related and previous work on reconfigurable computing architecture and briefly reviews the existing hardware and software scheduling algorithms. A simple overview of the target architecture is given in Section 3. The detailed description of the proposed CRS scheduling algorithm is presented in Section 4. Experimental results are given in Section 5 followed by conclusions.

2. Related work

2.1 Reconfigurable computing architecture

The concept of a reconfigurable computing has been around since the 1960s consisting of a standard processor and an array of "reconfigurable" hardware [2]. In the last decade there has been a recent renaissance in this area of research

3-9810801-0-6/DATE06 © 2006 EDAA

with many proposed reconfigurable architectures developed both in industry and academia such as, Matrix, Garp, Elixent, PACT XAPP, SiliconHive, Montium, Pleiades, Morphosys [3-10] to name but a few. These designs have only really become feasible by the relentless progress of silicon technology, allowing for complex designs to be implemented onto a single chip. Nevertheless, the development of reconfigurable hardware architecture is only one side of the problem, often overlooked is the importance of appropriate software tools that are essential for the programmability of the system and gaining the maximum performance. However, a lot of the industrial companies do not disclose their programming techniques; furthermore many reconfigurable systems focus on developing the hardware side while the implementation of algorithms to the architecture is manually carried out. This has an impact on the effectiveness and ease of programmability. Additionally reconfigurable computing cores tend to be developed in terms of flexibility and high amount of parallel processing with little focus on the potential power savings aspect of the design.

2.2 Scheduling algorithms

A number of scheduling algorithms have been developed for hardware high-level synthesis. As soon as possible (ASAP) and as late as possible (ALAP) scheduling use precedence constraints [12] but do not consider resource usage. List scheduling methods [13] are targeted for resource-constrained problems by identifying the available time instants in increasing order and scheduling as many operations at a certain instant before moving to the next. The integer linear programming (ILP) method [14] tries to find an optimal schedule using a branch-and-bound search algorithm, which is relatively straightforward to automate. However, most of these algorithms are based on the simplex method which is only applicable for very small problems. Force directed scheduling (FDS) [15] acts to minimise hardware subject to a given time constraint by balancing the concurrency of operations, the value to be stored, and data transfers. Iterative scheduling method [16] is a heuristic scheduling algorithm that permits escape from local minima. Each edge describes a precedence constraint between two nodes. The precedence constraint is an intra-iteration one if the edge has zero delays or inter-iteration if the edge has one or more delays [17]. All these algorithms concentrate only on the intra-iteration precedence constraints. The MARS scheduling algorithm [18] is a high-level DSP concurrent scheduling and resource allocation algorithm. It exploits both inter-iteration and intra-iteration precedence constraints. This scheduling algorithm implicitly performs algorithmic transformations such as pipelining and retiming, and is capable of generating valid architectures for algorithms that have randomly distributed delays.

A number of effective local and global scheduling algorithms are known for instruction level parallelism. These include critical path list scheduling [13], trace scheduling [19], and percolation scheduling [20]. However, all existing hardware scheduling algorithms do not consider operation chaining and the timing effects of reconfigurable function unit and routing interconnection delay. In addition, these software scheduling algorithms are limited to independent instruction parallelism and to a fixed number of registers. Therefore, there is a strong requirement for new compiler scheduling technique(s) to consider issues such as operation chaining, interconnection delay, power consumption and register assignment. This is the issue addressed in this paper.

3. Instruction cell based architecture

To meet the needs of future mobile multimedia systems that are limited in battery resources and must operate in a changing application and communication environment, a dynamic reconfigurable system architecture was presented in [11]. The architecture aims at simultaneously maintaining high-throughput while still staying efficient in terms of power consumption and silicon cost. Such an architecture is targeted at high performance and low power communication, embedded and portable applications.

The salient feature of the architecture is that both the programmable cells and their programmable interconnections can be dynamically reconfigured at specific points in time. In this architecture, the number and type of these cells are parameterisable upon application. The architecture, which is currently implemented on a UMC 0.13um CMOS technology library, consists of distinct 32-bit programmable functional units, general purpose registers, and an interconnection network. The basic and core elements of the architecture are the programmable cells which can be programmed to execute one type of operation similar to a CPU instruction. The ICs are interconnected through an island-style mesh architecture which allows operation chaining in the datapaths. The delay of each cycle is variable and programmable at run-time. Hardware feature are provided in the architecture to adapt the clock duty cycle.

An instruction cell based architecture has the same flexibility of coarse-grain FPGA and programmability of DSP. Since it employs a coarse-grained reconfigurable architecture, such an architecture has lower power consumption than generic fine-grained FPGA. A more detailed description of this architecture can be found in [11].

4. System level CRS scheduling algorithm

Traditional high-level synthesis (HLS) is a translation process which involves taking a behavioral description into a register transfer level (RTL) structural description. Scheduling is a critical task in this process. Scheduling partitions the Control Data Flow Graph (CDFG) into time steps thereby generating a scheduled CDFG model. After the scheduling routine is performed, register allocation is used to minimise the registers in the design. Differing from the traditional HLS

process, the proposed method combines the scheduling, routing, instruction cells binding and register allocation together to suit the instruction cell-based architectures. We present a new efficient multi-objective scheduling optimisation to give both high throughput performance and low power for new reconfigurable architectures. List scheduling (LS) is the most commonly used compiler scheduling technique for basic blocks. The main idea is to schedule as many operations as possible at a certain clock cycle before moving to the next clock cycle. This procedure continues until all operations have been scheduled. The major purpose of this way is to minimise the total execution time of the operations in the DFG. Of course, the precedence relations also need to be respected in list scheduling. All operations, whose predecessors in the DFG have completed their executions at a certain cycle t, are put in the so-called ready list R[t]. The ready list contains operations that are available for scheduling. If there are sufficient unoccupied resources at cycle t for all operations in R[t], these operations can be scheduled. However, if an appropriate resource for each operation in R[t] is not available, a choice has to be made on which operations will be scheduled at cycle t and which operations will be deferred to be scheduled at a later time. This choice is normally based on heuristics, each heuristic defining a specific type of list scheduling. By computing priories in a sophisticated way, they try to schedule the most critical cells first, and to assign them to the right resource. LS usually employ a priority vector to determine what tasks to consider first. In this paper, we present three approaches dealing with priority allocation mechanism and select the optimum one to generate the configurable bits for the target architecture.

Fig. 1 provides an algorithmic description of the new operation Chaining Reconfigurable Scheduling algorithm (CRS) with resource and time constraints. The traditional list scheduling can not be directly used on an instruction cell based architecture because: 1) an efficient operation chaining is required to deal with dependent instructions parallelism; 2) register allocation need to be considered in the scheduling algorithm; 3) The scheduling algorithm should also take the time effects of reconfigurable function unit and routing interconnection delay into account, which can change the data path delay in reconfigurable devices. To easily compare the CRS algorithm with simple list scheduling, the CRS algorithm consists of simple list scheduling algorithm as shown in Fig.1 with the additional new lines marked with a @.

Since the design entry in the CRS scheduling algorithm is the compiled and scheduled assembly code, the scheduler must handle the additional complexity such as removal of unnecessary registers. Register allocation is performed after scheduling in high-level synthesis. Since the scheduling and chaining of operations affect the register allocation, the CRS scheduling algorithm also considers the register assignment. The control flow graph is a fundamental data structure that is used to generate the correct data dependence between two

```
Algorithm CRS Scheduling
Input: Assembly Code representing operations to schedule,
       Space machine description (resource, clock cycle),
       Routing delay generated by VPR algorithm;
Output: Fast and parallel Netlist
Begin
Step 1: construct control flow graph (CFG)
Step 2: construct data flow graph (DFG)
Step 3: rename to eliminate anti/output dependences
Step 4: assign priority to instructions based on cells flexibility
Step 5: iteratively select & schedule instructions
Cycle =1;
Ready [Cycle] = Roots of DFG;
Ready [Cycle+1] =ϕ;
Scheduled [Cycle] =ϕ;
@Available_Register[Cycle]={the output registers but not used as input registers
    of basic block}
While (Ready Lists ≠ϕ) //All Ready[i] lists
{
 If (Ready [Cycle] =ϕ) then
 {
@   Initial available hardware resource;
     Cycle = Cycle + 1;
 }
 While (Ready [Cycle] ≠ϕ) {
@     Remove an op from Ready in priority order and a series low power
    optimisation techniques are used to select an op
   If (∃ free issue units of type(x) on cycle
@    && operation can be chained in the same cycle) {
      S (op) = Cycle;
      F (op) = FinishTime (op);
      Scheduled [Cycle] = Scheduled [Cycle] ∪ {op};
       For (each successor s of op in DFG){
          If (s is ready) then
              Ready [Cycle] = Ready [Cycle] ∪ s;
       }
  }
  else
       Ready [Cycle+1] = Ready [Cycle+1] ∪ {op}
 }
@If (op uses register and each successor of op in Scheduled [Cycle])
@then {//release register
   Remove Register that saves the output of op and Put it in available register
list}
@ N = the number of registers are needed in this clock cycle
@ A = the number of registers are available in this cycle
@ if (N<A)
@    Assign Register to save the output of op
@ else {
@     Remove some scheduled ops in terms of priority until N ≤A;
@     Assign Register to save the output of op;}
Θ Further scheduling using CPLS (this step is only used for MPLS)
@Calculate the longest critical path and variable clock cycles
}
@ Resource binding – using hamming distance.
End Algorithm
```

Fig.1 Operation Chaining Reconfigurable Scheduling Algorithm with Resource and Clock Cycle Constrained

basic blocks and to find the instruction parallelism for different basic blocks.

Three methods are adopted on the CRS scheduling algorithm, and the output code with the minimal number of clock cycles (highest throughout) or lowest power consumption is selected as scheduler's output. The first

method is called the critical-path reconfigurable scheduling (CPRS). In this context, the longest path from a node to an output node of the DFG is called its critical path. The maximum of all critical path lengths gives a lower bound value of the total time necessary to execute the remaining part of the schedule. Operations with higher priority have smaller mobility that is defined as the length of schedule interval (mobility = ALAP-ASAP). Therefore, in CPRS scheduling, nodes with the greatest critical-path lengths are selected to be scheduled at cycle t. The second method is called independent instruction priority reconfigurable scheduling (IPRS). In IPRS, nodes are selected to be scheduled at cycle t if they minimally increase the critical path of this clock cycle compared to other available nodes. Finally, the third method called mixed priority reconfigurable scheduling (MPRS) is also used in the CRS scheduler, which combines the CPRS and IPRS algorithm. In MPRS, nodes are firstly scheduled using IPRS method, and the CPRS algorithm is executed again after register allocation (A line marked with a Θ in Fig.1 is adopted only for MPRS). The CPRS algorithm will generate the minimal execution time for fixed clock system. Since the CRS algorithm adopts variable clock cycles, the CPRS scheduling method may result in longer execution time compared to the IPRS scheduling. As the CRS algorithm combines with register allocation, the finite amount of register may limit IPRS method. The MPRS scheduling may generate better scheduling than the CPRS and IPRS scheduling.

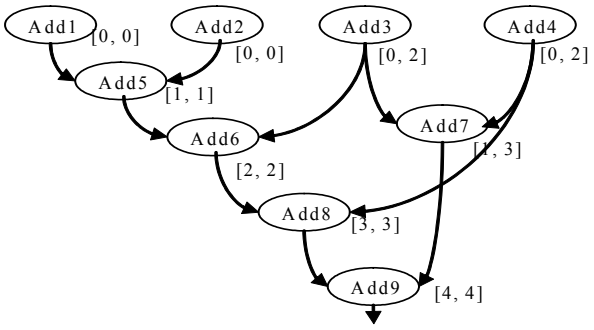

Fig. 2: Input Date Control Flow Graph

These scheduling techniques will be illustrated using the example shown in Fig.2. Assuming the architecture provides 5 operational elements that can execute 3xADD and 2xREG simultaneously and the addition is executed in one clock cycle, the ASAP and ALAP times for each adder are shown in square brackets in Fig.2. Table 1 compares three different CRS algorithm to see how they impact on the number of clock cycles needed to run Fig.2. function. In Table 1, ET refers to the total clock cycles of execution times (ET) and SC is scheduled cells. Table 1 shows that the CPRS scheduling generates longer ET than the IPRS scheduling, and the same ET as the MPRS scheduling. However, the ET of the different algorithms is heavily dependent on machine description. Tables 2 and 3 give the different scheduling results based on the different machine description. More shown in Table 1 and 2, the CPRS algorithm results in longer (Table 1) or shorter (Table 2) ET than the IPRS algorithm, which is caused by the number of registers. Otherwise, the IPRS algorithm generates the shorter ET time if there are enough registers (Table 3). As the functional resources only include adders in this example, the results using the MPRS algorithm is either the same as CPRS (Table 1-2) or the same as IPRS (Table 3). However, when applied to a greater variation in resource as in a reconfigurable instruction cell based machine, it generates different results. The compiler will schedule the input code using all methods and select an appropriate output code for their requirement.

Table 1: Comparisons of Schedulers (3 adders and 2 registers)

Execution Step	CPRS		IPRS.		MPRS	
	Cycles	SC.	Cycles	SC.	Cycles	SC.
1	2	[1,2,5]	1	[1,2]	2	[1,2,5]
2	2	[3,6]	1	[5,3]	2	[3,6]
3	2	[4,8,7]	1	[6]	2	[4,8,7]
4	1	[9]	2	[4,8,7]	1	[9]
5			1	[9]		
ET	7		6		7	

Table 2: Comparisons of Schedulers (4 adders and 2 registers)

Execution Step	CPRS		IPRS.		MPRS	
	Cyc.	SC.	Cyc.	SC.	Cycles	SC.
1	2	[1,2,5,3]	1	[1,2]	1	[1,2,5,3]
2	2	[6,4,8,7]	2	[5,3,6]	1	[6,4,8,7]
3	1	[9]	3	[4,8,7,9]	2	[9]
ET	5		6		5	

Table 3: Comparisons of Schedulers (5 adders and 5 registers)

Execution Step	CPRS		IPRS.		MPRS	
	Cyc.	SC.	Cyc.	SC.	Cyc.	SC.
1	3	[1,2,5,3,6]	2	[1,2,3,4,5]	2	[1,2,3,4,5]
2	3	[4,8,7,9]	3	[6,7,8,9]	3	[6,7,8,9]
ET	6		5		5	

For each basic block, a data flow graph (DFG), which represents data dependence among the number of operations, is used as an input to the CRS scheduling algorithm. One limitation of running a design on processors is the number of operations that can be executed in a single cycle. To maximise the number of operations executed in a single cycle, operation chaining and variable clock cycle are added to the CRS scheduling algorithm. It reduces the total number of registers usage, decreases the power consumption by reducing memory access, and increases throughput by variable clock cycles. When chaining an operation, scheduling algorithms must consider not only the operation's impact on the critical path but also the programmable routing delay. The scheduling algorithm in the tool flow includes pre-scheduling and post-scheduling. The pre-scheduling algorithm considers the operation and estimated routing time effects on the critical

path. The Netlist after pre-scheduling will be fed into a standard placement and routing tool VPR [22]. The corresponding routing information given by VPR tool is used as input to post-scheduling algorithm in order to generate the Netlist that met design requirement. The CRS algorithm is used in the first phase for performance optimization. After that, the resource binding is generated by using Hamming distance as our power cost model to estimate the transition activity in instruction cell configuration bus [23]. Hamming distance is the number of bit differences between two binary strings. The resource binding scheme with the minimum Hamming distance is chosen to reduce the power consumption.

5. Experimental Results

Benchmark tests are conducted using different CLS scheduling algorithms (CPLS, IPLS and MPLS) and are targeted on the same reconfigurable instruction cell based architecture [11] to demonstrate the performance of each CLS algorithm. Table 4 shows the execution time and energy consumption where the benchmarks are running on the architecture at the same frequency (125MHz) and the same hardware resources are used. The power consumption values for our reconfigurable architecture are obtained after post-layout simulation by the Synopsys PrimePower using UMC 0.13um technology.

Table 4: Comparison of Different Scheduling Method

Benchmark	Method	Execution Times (us)	Energy (uJ) consumption
2D-DCT	CPLS	2.352	0.1309
	IPLS	2.192	0.108
	MPLS	2.192	0.112
FIR	CPLS	1.998	0.263
	IPLS	1.98	0.248
	MPLS	1.98	0.249
IIR	CPLS	0.194	0.015
	IPLS	0.186	0.0154
	MPLS	0.188	0.0156
MinError	CPLS	9.286	1.59
	IPLS	9.086	1.519
	MPLS	9.286	1.529
OFDM	CPLS	42.496	0.7980
	IPLS	43.224	0.8003
	MPLS	41.6	0.7905
Viterbi	CPLS	501.488	0.2296
	IPLS	452.162	0.225
	MPLS	452.1	0.227
Dhrystone	CPLS	283.046	1.004
	IPLS	281.08	0.878
	MPLS	282.06	0.925

From the table, we can see that for all the benchmarks the CPLS, IPLS and MPLS algorithms achieve slightly different execution times and power consumption values. Here, scheduled code can be selected by the end user's design criteria, e.g. power, throughput. In order to provide the high throughput, the output code with the lowest execution time is selected. The optimal solution is heavily dependent on applications, it may be generated by CPLS, IPLS or MPLS algorithm. This has been illustrated in Section 4 (Table 1-3). In most benchmarks, the CPLS scheduling algorithm provides less register usage but however it has higher energy consumption compared to other scheduling algorithms. From scheduling and simulation analysis, the code generated by the CPLS scheduling algorithm has longer combinational logic path than other algorithms, which results in high power consumption. However, the CPLS algorithm provides less energy consumption in some cases. Once again, energy consumption is dependent on applications. The compiler will schedule the input code using all methods and select an appropriate output code for their requirement. The CLS scheduling algorithm provides the potential lower energy consumption and a similar throughput compared to others DSP/VLIW processor [11].

6. Conclusions

In this paper we have presented a new scheduling algorithm (CRS) targeted for an instruction cell based reconfigurable computing architecture. Unlike other scheduling methods, it considers the placement, routing effect, register assignment, resource binding and advanced operation chaining issue found in new distributed reconfigurable computing architectures, allowing efficient implementations of complete high performance system solutions. Future work will deal with extending the scheduling technique to include instruction pipelining and code size minimization.

7. References

[1] Power Comparison, RapidChip® Platform ASICs vs. FPGAs

[2] G. Estrin, "Organization of Computer Systems -The Fixed Plus Variable Structure Computer," Proc. Western Joint Computer Conf., New York, 1960, pp. 33-40

[3] E. Mirsky and A. DeHon, "Matrix: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources" IEEE symposium on FPGAs for custom computing machines, April 1996, pp.157-166.

[4] J.R. Hauser "Augmenting a Microprocessor with Reconfigurable Hardware" Thesis, University of California, Berkeley, 2000

[5] D-Fabrix processing array, Reconfigurable Signal Processor, www.elixent.com, 2004

[6] XPP, PACT, "OFDM decoder for wireless LAN – whitepaper" www.pactcorp.com, May 2002

[7] Reconfigurable Computing, Philips, Avispa, www.siliconhive.com, 2004

[8] P.M. Heysters, G.J.M Smit and E. Molenkamp, "Montium – Balancing between Energy-Efficiency, Flexibility and Performance" Engineering of Reconfigurable Systems and Algorithms, 2003, pp.235-241.

[9] M. Wan, H. Zhang, V. George, M.Benes, A. Abnous, V. Prabhu and J.Rabaey, "Design Methodology of a Low Energy Reconfigurable Single-Chip DSP System" Journal of VLSI Signal Processing, 2000, pp. 53-63.

[10] H. Singh, M.-H. Lee, et al. "Morphosys: an integrated reconfigurable system for data-parallel and computation-intensive applications". IEEE Trans. on Comp., vol49(5):, May 2000, pp. 465–481.

[11] Reconfigurable Instruction Cell Array, U.K. Patent Application Number 0508589.9.

[12] P. DeWilde et al., "Parallel and Pipelined VLSI Implementation of Signal Processing Algorithms", in VLSI and Modern Signal Processing, eds. Kung, Whitehouse, and Kailath, Prentice Hall, 1985.

[13] S. Davidson, D. Landskov, B.D. Shriver, P.W. Mallett, "Some Experiments in Local Microcode Compaction for Horizontal Machines", IEEE Trans. On Computer, vol. 30, no. 7, 1981, pp. 460-477.

[14] J. Lee, Y. Hsu, and Y. Lin, "A new Integer Linear Programming Formulation for the scheduling Problem in Data-Path Synthesis," Proc. of the Int'l. Conf. on Computer Aided Design, 1989, pp. 20-23.

[15] P. G. Paulin and J. P. Knight; "Force Directed Scheduling For the behavioral synthesis of ASIC's," IEEE Trans. On Computer Aided Design, Vol.8, June 1989, pp.661-679.

[16] I-C. Park and C-M Kyung, "Fast and Near Optimal Scheduling in Automatic Data Path Synthesis," Proc. of the 28th DAC, 1991, pp. 680-685.

[17] K. K.Parhi, "VLSI Digital Signal Processing Systems Design and Implementation", John Wiley & Sons, Inc., 1999.

[18] C-Y Wang and K. K.Parhi, "The MARS High-Level DSP Synthesis System", VLSI Design Methodologies for Digital Signal Processing Architectures, Kluwer Academic Publishers, 1994.

[19] J.A. Fisher, "Trace Scheduling: A Technique for Global Microcode Compaction", IEEE Trans. On Computers, vol. 30, no. 7, 1981, pp. 478-490.

[20] A. Aiken, A. Nicolau, "A Development Environment for Horizontal Microcode", IEEE Trans. On Software Engineering, no. 14, 1988, pp. 584-594.

[21] M.W. Hwu, R.E. Hank, D.M. Gallagher, A.M. Scott, D.M. Lavery, G.E. Haab, J.C. Gyllenhaal, and D.I. August, "Compiler Technology for Future Microprocessors", Proceeding of the IEEE, 83(12), December 1995, pp.1625-1639.

[22] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research", in Proc., FPL 1997.

[23] Chingren Lee, Jeng Kuen Lee, TingTing Hwang and Shi-Chun TSAI, "Compiler Optimization on VLIW Instruction Scheduling for Low Power", ACM Transactions on Design automation of Electronic Systems, Vol. 8, No. 2, April 2003, Pages 252-268.