3rd International Symposium on Communications and Information Technologies, 3-5 September 2003, Thailand

Hardware multiplexing: towards a resource efficient reconfigurable processor

K. Piromsopa, P. Bavonparadon, P. Chongstitvatana
Department of Computer Engineering
Chulalongkorn University
Phayathai road, Bangkok, 10330, THAILAND
E-mail: prabhas@chula.ac.th

ABSTRACT

This work proposes an architecture that achieves efficient and flexible use of resource. This architecture is based on reconfigurable capability. The hardware-multiplexing scheme makes use of limited resource and achieve performance by fine grain parallelism using data flow approach. It is suitable for the future mobile devices, which require both low power and high performance. The preliminary experiment shows that there exists a design space in which the proposed architecture is 3-4 times faster than a fully pipeline processor with a comparable resource. The constraint of the speed of reconfiguration is reported.

1. INTRODUCTION

Hand held devices are becoming prevalent in today society. Mobile phone, PDA and the like are increasingly more powerful, they will play media rich data: songs, movies and will be the center of communication for everyday life. They require powerful processors to perform their function at the same time the power consumption is most stringent constraint. The research into low power electronics, at every level: system level, architectural level, circuit level and electronic level are crucial.[1]. At architectural level, the media rich application has driven the design into a new direction [2] that is suitable for fine grain parallelism.

One of the new technology that emerges recently and develops rapidly to become highly competitive with ASIC (Application Specific IC) technology in realizing computational task is the field programmable gate array (FPGA). Initially the FPGA has been used for a rapid turn around time low volume implementation of a digital design. However as the speed and the density of FPGA devices increase, it is possible to realize complex design such as an advanced processor in one chip. Custom Computing Engine becomes possible and has been explored in many applications [3].

The motivation in exploring the possible design space of FPGA technology for the future hand-held applications arises from the need to meet both criteria of low power and high performance. Although these two requirements seem to be in the opposite direction, we believe that some possible design exists with the reconfigurable capability of the future FPGA devices. This work explores the hardware multiplexing as a new architecture in this design space. For fine grain parallelism, an application program is compiled into a data flow graph. A hardware that realizes this computation can be synthesized. To implement this circuit with the constraint on resource, the data flow graph is partitioned and is configured into a FPGA device to be executed step by step. Hence, the hardware is time-multiplexed on the FPGA device according to the partitioning of the data flow graph. This scheme of "hardware multiplexing" enables a reconfigurable device to realize a full computation of data flow graph with limited resource.

The paper is organized as follows. The next section describes the related work. In Section 3, we look at the scheme of hardware multiplexing in more details. The experiment comparing this architecture with the conventional architecture is demonstrated in Section 4. Finally, in Section 5 we discuss the result of the experiment and its implication on the future FPGA devices.

2. Related Work

There are tremendous amounts of work done on reconfigurable processors. Many of which are the extensions of traditional processors to handle multimedia applications [4, 5, 6]. The reconfigurable part becomes a co-processor to accelerate multimedia operations.

In [4], the paper presents a relatively simple processor with a dynamically reconfigurable data path acting as an accelerating co-processor. This data path consists of hardwired function units and reconfigurable interconnect. The FPGA is configured as a co-processor only in the initiation phase. Furthermore, the hardwired function unit is fixed during the execution phase, and the reconfigurable interconnections are controlled by the controller in the co-processor for changing the interconnection inside This work reconfigures both the function unit and the interconnection very often during the execution. Therefore, our work is more flexible than this. Moreover, our data flow machine should be faster than a processor with a co-processor since there is no communication overhead.

The [5] studies the performance of the reconfigurable co-processor on multimedia applications. This work uses the similar approach as in [4] that the system is composed of a processor and a reconfigurable co-processor. It compares an FPGA based reconfigurable co-processor with a reconfigurable array processor. The results show that the FPGA co-processor needs more hardware area to achieve the same performance improvement, as the reconfigurable array processor needs. This research is a good suggestion of the technology, which may be suitable in our work.

In [6], it proposes and evaluates a programmable loop engine (PLE) that executes media codes and kernels efficiently by moving most of the overhead associated with media program into hardware. The PLE has a similar concept as the reconfigurable co-processor of [4,5]. This work compares the PLE and the processor with SIMD extension. The result shows that the PLE requires less hardware resource than the SIMD extension, and the PLE is also faster.

The [7] and [8] study the partitioning methodology for a reconfigurable processor. The goal of the methodology is to minimize the reconfiguration overhead. In [9], it presents a synthesis methodology, which starts from high-level system specifications and synthesizes run-time reconfigurable systems. Furthermore, the [10] proposes a scheduling method to determine static operation execution time and function unit allocation to achieve fast signal processing by considering dynamic reconfiguration of function units. These works provide the algorithms and methodologies supporting the reconfigurable systems.

The proposed architecture is different from the existing work as it emphasises on the multiplexing of computational nodes subject to a limited resource. The whole computation is partitioned according to the available resource with reconfiguration plays the part of time multiplexing. This allows trade off between resource and performance at different dimension.

3. Hardware multiplexing

To describe an idea of a design it is easier to begin with a concrete example. The illustrated program computed prime numbers using a sieve of Eratosthenes method. The array a[x] stored the flag 0,1 indicates whether x is prime, initially a[.] = 1. This program iterates to find all primes less than MAX.

```
        void sieve()   {
        int p, j;
1       p = 2;
2       while ( p*p <= MAX ) {
3          j = p + p;
4          while ( j <= MAX ) {
5             a[j] = 0;
6             j = j + p;
           }
7          p = p + 1;
8          while( a[p] == 0 )
9             p = p + 1;
           }
        }
```

The data flow graph of this program is shown in Figure 1 where each node of the graph represents each line of the program (the last while loop the line 7 and 9 are folded).

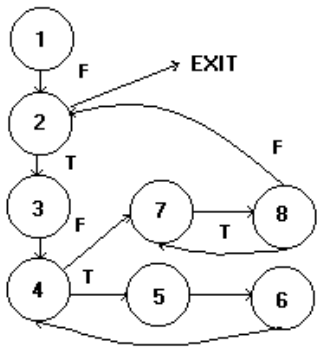

Figure 1 the data flow graph of the sieve program

Each node is realized by direct synthesis of operators, variables and constants. A variable is stored in a register. The variables are p and j. The constants are the address of a[] and MAX. The computation nodes are shown in Figure 2.

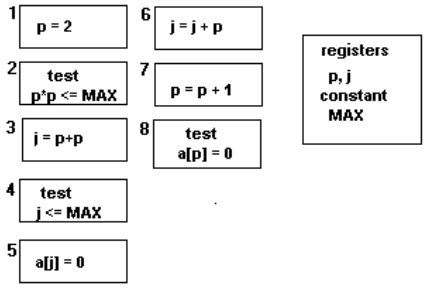

Figure 2 the computation node of the sieve program

After initialization phase of line 1, the graph is "executed" step by step until the computation is terminated. This computation can be implemented on a limited resource reconfigurable device by partitioning this graph. Let us assume the graph is partitioned into individual node, then the graph can be executed one node at each step, the intermediate results are stored on persistent registers and the computing unit is reconfigured to be the next node. This enables the whole computation to be multiplexed into a limited resource hardware.

The reconfigurable processor (RC) is composed of three parts: registers, a control unit and a reconfigurable unit. The registers are persistent and store the intermediate results among the different configurations. The control unit is static, it holds the computation states and activates the reconfiguration. The reconfigurable unit realizes a partition of the data flow graph and is reconfigured according to the state of computation.

4. Evaluating the architecture

The proposed architecture is evaluated by comparing the speed and the required resources with a simple 32-bit load/store processor without pipeline implementing on the same technology (FPGA). The S2 processor is used for teaching purpose in our department [12]. Figure 3 shows S2 architecture.

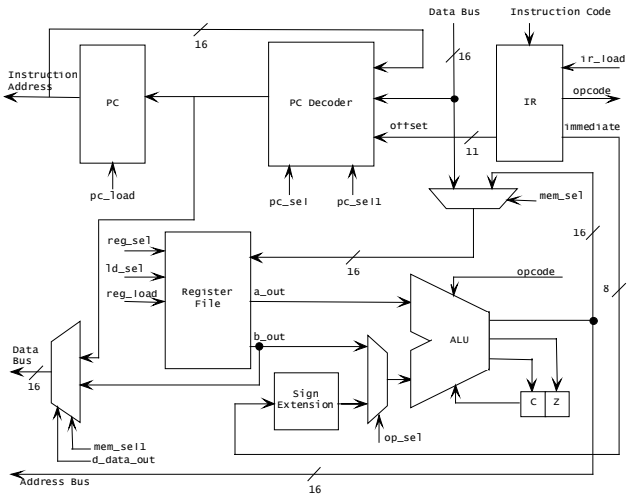

Figure 3 the architecture of S2 processor

The questions we want to ask are

1) Given a fixed amount of resource how does RC perform against S2?

2) How fast and how often RC needs to be reconfigured to achieve a level of performance?

The preliminary study evaluates only the integer arithmetic and uses the small benchmark suite [11]. The benchmark programs consist of seven programs (Table 1):

Prog.	Description
Sieve	Find prime number <= 100 by the method of Erathothenes
Hanoi	Move 6 disks from peg 1 to peg 3
Matmul	Multiply 4x4 matrix
Bubble	Bubble sort, input 20..1
Qsort	Quick sort, input 20..1
Perm	Permutation generator, permute 4 numbers: 0 1 2 3
Queen	Find all solutions of 8-queen problem encoding the solution as column position {0,1,2,3,4,5,6,7}

Table 1 the benchmark suite and input

In term of performance, the measurements are the dynamic instruction count of S2, the number of time step required by the proposed architecture (RC) and the number of reconfiguration of RC. We are not interested in the absolute speed as it does not contribute to our understanding about the reconfiguration constraint. The amount of resource required by both processors are compared. The resource for S2 is constant. The resource for RC is variable as it depends on the size of reconfigurable unit. For this experiment, it is fixed to the amount necessary to realize about 3 nodes of the data flow graph. The partitioning of the data flow graph is done manually aiming to minimize the amount of reconfiguration. To make a reasonable comparison, we further assume that each instruction of S2 takes the same amount of time. Table 2 shows the statistics of both processors running the benchmark suite. "RC step" is the number of time step required by RC to execute the benchmark. "RC con" is the number of reconfiguration of RC. "S2 inst" is the number of instruction executed by S2. "Step/Con" indicates how often the reconfiguration is required compared to the computation. "Inst/Con" is similar but compares the reconfiguration of RC to the computation of S2. "Inst/Step" is the speed up of RC over S2 if it is assumed that one step of RC is equal to the time for one instruction of S2 (this is an underestimate for RC). The discussion of these results is in Section 5.

To measure the resource for both processors, both processors are synthesized using Xilinx web pack targeting XC2s100 (a 100,000 gates Spartan II device). The resources required by S2 are shown in Table 3 in terms of the equivalent gate.

Prog.	RC Step	RC con	S2 inst	Step/ con	Inst/ con	Inst/ step
Sieve	380	13	1607	29.23	123.62	4.23
Hanoi	1494	187	3637	7.99	19.45	2.43
Matmul	621	249	2229	2.49	8.95	3.59
Bubble	2912	951	14549	3.06	15.30	5.00
Qsort	1227	302	4640	4.06	15.36	3.78
Perm	1878	715	6264	2.63	8.76	3.34
Perm5	1878	130	6264	14.45	48.18	3.34
Queen	115599	54669	855001	2.11	15.64	7.40

Table 2 Performance comparison of RC and S2

Unit of S2	Equivalent. Gate
MUX 2:1	384
ADDER	378
ALU*	3189
IR	256
PC	256
Registers 32x32	16,678
Control unit	4,000
Total S2	25,141
Total RC	18,978

Table 3 the resource required by S2 and RC

Please note that the multiply and divide instructions of S2 are not implemented, similarly for the RC processor. For the RC processor, the control unit is variable and the required resource is approximately 50 gates per state. The largest program in the benchmark suite is 8-queen and it has 16 states. The register file is the same as S2. The reconfigurable unit is fixed at 1,500 gates. The total amount of resource for RC to run all the benchmark is 50*16 + 16,678 + 1,500 = 18,978 gates. (control + registers + reconfigurable gates) The following table shows an example of the amount of resource required for the sieve program for RC. For the partition of {1,{2,3},{4,5,6},{7,8}} the reconfigurable unit required to hold the largest partition {4,5,6} is 1,020.

Nodes of RC	Equivalent. gate
Registers 2	1024
state 1	118
state 2*	378
state 3	118
state 4	310
state 5	214
state 6	496
state 7	310
state 8	198
control unit	366

Table 4 The resource for each node of the sieve program

(Please note that the multiplier is not implemented in state 2 and an adder is reported instead)

Each state of RC does not have to run on the same clock. Thus, clock frequency may differ in each step. The control unit issues start command to reconfigurable unit and wait for stop signal. This schema allows system to be implemented as either synchronous or asynchronous sequential circuit.

5. Discussion

First, we will discuss the resource comparison. Looking at Table 3, the comparison of the resource is not quite differentiable as the number is overshadowed by the amount of resource used by the register file. However, it can be said that RC is not larger than S2. To illustrate the possible advantage of the reconfigurable unit, let us compare S2 with the sieve program of RC with two registers and S2 with four registers.

S2 runs sieve with (4 registers) = 384+378+3189+256+256+(2084)+4000 = 10,547

RC runs sieve with (2 registers) = 1024+366+1020 = 2,410 (register + control + reconfigurable)

In this comparison, RC uses about 5 times less resource than S2 (or it uses only 20% of the resource of S2).

To answer the performance question, if the resource is assumed to be similar as in Table 3 and the reconfiguration time is ignored, then the speedup of RC over S2 is ranging from 2.43 to 7.4 (Table 3 Inst/Step). This is the amount of speedup or the gain of RC over S2 when they are clocked at the same rate.

Lastly, we will discuss the constraint on reconfiguration. The frequency of reconfiguration varies depended on the partitioning. In the worst case, it is about 2 steps per reconfiguration. In the best case, it is 29 steps per reconfiguration. The best case is achieved by putting the whole loop in the same partition ({4,5,6} in the sieve program). To illustrate this point, another example of different partition is shown in Table 2 for perm5. The partition in perm required 715 reconfigurations but perm5 which partition into 2 big partitions (larger than our limit of 1,500 gates per partition but this is for illustrative purpose) can reduce the amount of reconfiguration to only 130 and the step/con are increased from 2.63 to 14.45. This shows that partitioning is critical. The amount of reconfiguration is the loss of performance of RC over S2. The gain is 3-4 instructions per step because of fine grain parallelism and the loss is the time required for reconfiguration. The frequency of reconfiguration depends on the partition, which in turns depends on the resource available on the reconfigurable unit and the data flow graph. How fast the reconfiguration should be is constraint by the balance of these gain/loss factors. The preliminary data shows that the reconfiguration is very frequent possibly every two steps of data flow execution.

6. Conclusion

This work explores a new design space created by the capability of reconfigurable devices. The aim is to create an architecture that is suitable for future mobile devices, which require both low power and high performance. The proposed architecture has flexibility in terms of resource and performance. By using the scheme of hardware multiplexing, the resource can be used efficiently (in terms of area), and at the same time the performance is achieved using fine grain parallelism. The preliminary experiment reported here shows that this is possible. The proposed architecture can achieve 3-4 times speedup over an ordinary processor with a cycle per instruction equals 1.0 (a typical fully pipelined processor). There is a clear indication that the resource is used efficiently. Our future work is to probe further into the kind of reconfiguration technology and to find out more about the high performance reconfiguration necessary to make it possible to use hardware multiplexing.

7. REFERENCES

[1] S. Devadas, S. Malik, A survey of optimization techniques targeting low power VLSI circuits, in Proc. of the 32nd ACM/IEEE conference on Design automation conference, 1995 , San Francisco, California, United States, pp. 242 - 247

[2] C. E. Kozyrakis and D. A. Patterson, “A New Direction for Computer Architecture Research,” IEEE Computer, pp. 24-32, Nov. 1998

[3] R. Gonzalez, “Xtensa: A Configurable and Extensible Processor,” IEEE Micro, 20(2), March/April 2000.

[4] Z. Huang and S. Malik, “Applications of reconfigurable computing: Exploiting operation level parallelism through dynamically reconfigurable datapaths”, Proceedings of the 39th Design Automation Conference, USA, 2002.

[5] T. Miyamori and K. Olukotun, “A Quantitative Analysis of Reconfigurable Coprocessors for Multimedia Applications”, Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines, 1998.

[6] D. Talla, L. K. John and D. Burger, “Hardware Support to Reduce Overhead in Fine-Grain Media Codes”, Technical Report LCA-TR-011101, Laboratory for Computer Architecture, The University of Texas, Austin, 2001.

[7] J. Noguera and R. Badia, “A HW/SW partitioning algorithm for dynamically reconfigurable architectures”, Proceedings of the conference on Design, Automation and Test in Europe , 2001.

[8] S. Ganesan and R. Vemuri, “An integrated temporal partioning and partial reconfiguration technique for design latency improvement”, Proceedings of the conference on Design, Automation and Test in Europe, 2000.

[9] K. Rath and J. Li , “Synthesizing Reconfigurable Sequential Machines Using Tabular Models”, Proceedings of the 5th Reconfigurable Architectures Workshop (RAW-98), Orlando, Florida, 1998.

[10] K. Ito, “A scheduling and allocation method to reduce data transfer time by dynamic reconfiguration”, Proceedings on the 2000 conference on Asia and South Pacific design automation, 2000.

[11] J. Hennessy and P. Nye, "Stanford Integer Benchmarks", Stanford University.

[12] P. Chongstitvatana, “S2 processor and its opcode format,” http://www.cp.eng.chula.ac.th/faculty/pjw/teaching/ads/