Design of a GPU-styled Softcore on Field Programmable Gate Array

Nattapong Thammasan
Department of Computer Engineering
Chulalongkorn University
Bangkok 10330, THAILAND
Nattapong.Th@Student.chula.ac.th

Prabhas Chongstitvatana
Department of Computer Engineering
Chulalongkorn University
Bangkok 10330, THAILAND
prabhas@chula.ac.th

***Abstract* - This work describes a softcore design of a processor in style of Graphic Processing Units. The design is realized using Verilog Hardware Description Language. The proposed design has an advantage in the flexibility to scale up by adding more processing elements to attain more speedup. The design of its instruction set architecture is explained. A realization of four processing elements processor is presented. It requires 268,637 equivalent gates. The maximum frequency is 117 MHz. It is suitable of embedded applications. In term of cycles consumed, it compares very well to a test program running on commercial Intel's CPU, Core2 Duo P8400.**

Keywords-softcore design; graphic processing unit; FPGA;

I. INTRODUCTION

In recent years, difficult computational problems tend to be solved by hardware implementation to achieve high speed [1]. Another approach is to exploit parallel execution from Graphic Processing Units (GPUs) [2]. A hardware implementation, however, is very specific and cannot be applied to other problems. The use of GPU is much more flexible. The work reported here presents a design of parallel architecture in the style of Single Instruction Multiple Data stream (SIMD) architecture similar to a GPU. The design is realized on Field Programmable Gate Array devices. The main purpose of this hardware is to solve computational problems, not to run graphic tasks like what a GPU mainly does.

The remaining sections are organized as follows. Section 2 introduces the characteristics of GPU. Section 3 describes the proposed hardware design. Section 4 presents a synthesis result from the design. Section 5 concludes the paper.

II. CHARACTERISTIC OF GPU

The most important characteristic of GPU is its single instruction multiple data model. It can execute single instruction by multiple processing elements simultaneously. In the task that multiple data can be performed simultaneously by the same instruction, this kind of data parallelism can accelerate computation and give speedups. A modern GPU has a number of processing elements in one pipeline. Fig. 1 shows an example of a commercial GPU.

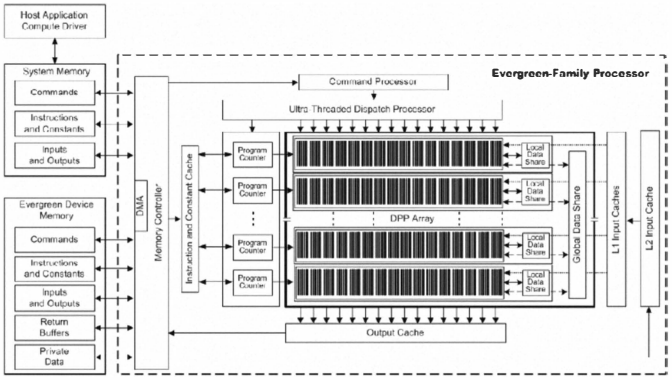

Figure 1. Evergreen Family GPU Block Diagram [3]

For example, Evergreen Family GPU of AMD [3] consists of data-parallel processor (DPP) arrays, a command processor, a memory controller, program counters, data sharing units, caches, and other logic. The DPP array is organized as a set of compute unit pipelines that operate in parallel on streams of data. The pipelines can process data or transfer data to, or from, memory. The programs for this GPU consist of instructions that operate on 32-bit or 64-bit IEEE floating-point values and signed or unsigned integers. Both graphic programs and general-computing applications use these instructions.

III. HARDWARE DESIGN

The proposed design is a 32-bit processor with no pipelined execution. This design is based on the SIMD model of a GPU. The hardware organization and the instruction set architecture are described in the following subsections.

A. Hardware Organization

Fig. 2 demonstrates a system overview of the proposed design. The processor has 32-bit data width and 10-bit address width. The degree of parallelism of hardware implementation depends on the number of processing elements. The processing elements are the most important modules that can execute basic operations in parallel. Other units such as the control logic and data memory are simplified so that the whole design can be synthesized into a compact device.

The hardware organization is shown in Fig. 2. Each major component will be explained next.

978-1-4673-1921-8/12/$31.00 ©2012 IEEE

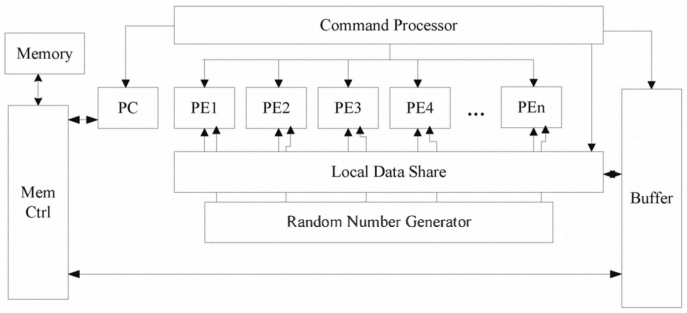

Figure 2. Hardware Organization

Memory Controller (MemCtrl) is the module to connect to system memory and local memory.

Program Counter (PC) is the module to indicate the current position of program.

Command Processor is the module to memorize the instruction fetched from a system memory. It also is connected to the processing elements, local data shared and buffer, so that the operand of some instructions can specify the addresses of units of each module to operate. For example, the operand of the instruction specifies the general purpose registers.

Buffer is the module to hold data receiving from, or sending to, a memory controller. It consists of registers of processing elements.

Local data share is the module to transfer data to, or from, each processing element, where each element can exchange their data to others. It is connected to buffer and it consists of registers of processing elements in one pipeline.

Random is the module to generate 32-bit random numbers for each processing element by exploiting linear feedback shift register.

Processing Element (PE) is an element to calculate arithmetic and logic operations. It consists of array of general purpose registers, the result register and an accumulator register. The hardware organization of processing element is shown in Fig.3.

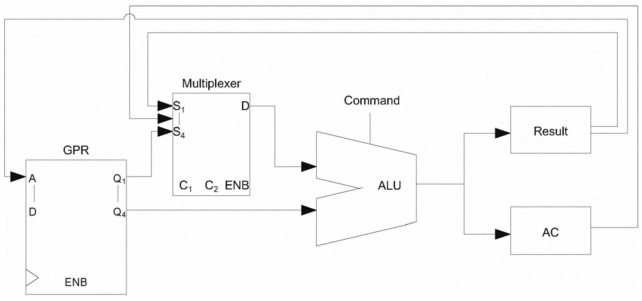

Figure 3. Hardware Organization of processing element

As we implement this design by using Verilog Hardware Description Language, we can scale up this design duplicating processing element module and connecting link into those modules correctly. We can do this because all processing element modules are not different form each other. In other words, they process the same instruction given by command processor simultaneously.

B. Instruction Set Architecture

The instruction set has a fixed length of 32 bits. Each instruction consists of two parts: 10-bit opcode and 22-bit operand. The instructions can be divided into four categories: arithmetic and logic, data manipulation, control, and others (See Table I). Fig.4 shows the example of instructions. A unary instruction operates on general purpose registers or other registers within the processing element. A binary instruction needs two operands, which are depended on the addressing mode. It can be either an immediate value or a local variable stored in the memory.

Normally, the processor takes;

- two cycles for an instruction fetch because the processor need to wait for memory read
- three cycles for an operand fetch those are one cycle to send data from memory controller to command processor, one cycle to decode addressing mode, and one more cycle to decode instruction
- one more cycle for executing the instruction
- For some complex instructions, the execution needs one or more cycles to complete. For example, the instruction that needs to keep its computational result into another general purpose register again can not do so directly. It needs to keep it into the result register for one cycle and send such data from the result register into specific general purpose register for one cycle. For store buffer instruction, the execution needs two cycle for sending data from buffer to memory controller and two more cycle to store data from memory controller to system memory or local memory.

As a result, an overall performance of this design is around 6 to 7 cycles per instruction.

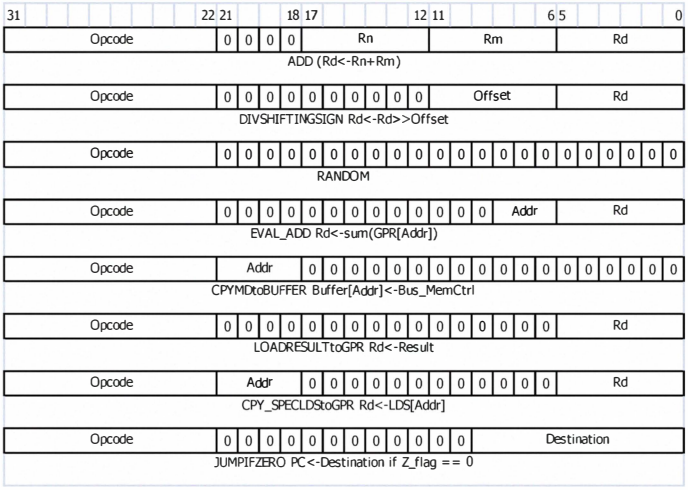

Figure 4. Example of Instructions

TABLE I. THE LIST OF THE INSTRUCTIONS OF THE PROPOSED DESIGN

Instruction category	Instruction
Arithmetic & Logic	ADD, SUB, MUL, DIV SHIFTING SIGN, INC, DEC, RANDOM, COMPARE, COMPARE_EQUAL, EVAL_ADD, CDTADD, ADDGPRAC_BOUND
Data Manipulation	CPYMDtoBUFFER, CPYBUFFERtoLDS, CPYLDStoBUFFER, CPYLDStoGPR, CPYPROCtoLDS, CDT_LOAD_GPR , LOADRESULTtoGPR, CPY_SPECLDStoGPR, STRBUFFER, CPYGPRtoLDS,
Control	JUMPIFZERO, JUMPIFNOTZERO, JUMPIFTRUE, JUMPIFNOTTRUE
Misc	NOOP

IV. MEASUREMENTS

In this section the processor is tested with three programs to validate the correctness of the design.

To give a picture of the size and performance of the design, it is compared to a well-known commercial softcore, Micro Blaze [4]. The test programs are: a matrix multiplication, the calculation of Mandelbrot set, and lastly, an application. A compact Genetic Algorithm ([5]) is implemented to illustrate the application.

The matrix multiplication C = A×B of two matrices A and B is conformable, if the number of columns of A is equal to the number of rows of B. The ijth element of C is given by

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

where, A = $[a_{ij}]$, B = $[b_{ij}]$ and C = $[c_{ij}]$ are matrices of appropriate dimensions.

From the study of parallel matrix multiplication, the nature of problem is proper to be solved in parallel by assigning each row and each column of matrix to each processor. From the equation above, cij is assigned to one processing element to compute the result. Fig.5 shows pseudo code of the matrix multiplication.

```
procedure MatrixMultiplication(A, B)
   input A, B n*n matrix
   output C, n*n matrix
begin
   for ( i = 0; i < n; i++)
      for ( j = 0; j < n; j++)
         for( k = 0; k < n; k++)
            C[i,j] = C[i,j] + A[i,k] * B[k,j]
         end for
      end for
   end for
end MatrixMultiplication
```

Figure 5. Pseudo code of matrix multiplication

As we compute 4x4 matrix multiplication by 4 processing elements, we unroll the for-loop fully so it becomes a "straight line" code consisted of 172 instructions. We assign each processing elements to compute each element in the result matrix row by row. From the pseudo code, each i-th processing elements is responsible to compute C[0,i], C[1,i], C[2,i], and C[3,i] for n=4. Fig.6 shows straight lines code of instructions in the part to compute C[0,i].

```
ld 0 @512
ld 1 @513
ld 2 @514
ld 3 @515
lds
ldr 0
ld 0 @516
ld 1 @517
ld 2 @518
ld 3 @519
lds
ldr 1
ld 0 @520
ld 1 @521
ld 2 @522
ld 3 @523
lds
ldr 2
ld 0 @524
ld 1 @525
ld 2 @526
ld 3 @527
lds
ldr 3 ;  load v0 to r0..r3
```

```
ld 0 @528
ld 1 @528
ld 2 @528
ld 3 @528
lds
ldr 4
ld 0 @529
ld 1 @529
ld 2 @529
ld 3 @529
lds
ldr 5
ld 0 @530
ld 1 @530
ld 2 @530
ld 3 @530
lds
ldr 6
ld 0 @531
ld 1 @531
ld 2 @531
ld 3 @531
lds
ldr 7 ; load a row v1' to r4..r7
```

```
mul 8 0 4
mul 9 1 5
mul 10 2 6
mul 11 3 7
add 12 8 9
add 13 10 11
add 14 12 13 ; inner product
str 14
buf
st @640 0
st @641 1
st @642 2
st @643 3 ; finish a row
(0)
;;
```

Figure 6. straight lines code of instructions in the part to compute first row of the result of matrix multiplication

Fig.7 shows pseudo code of the Mandelbrot set. The number of pixels that can be computed simultaneously depends on the number of processing elements per one pipeline. Four processing elements are assigned to compute the color of four pixels at the same time. As each processing element performs the same instruction, it is impossible to get out of the loop when the condition of one processing element is false. In other words, all processing elements must be synchronized to terminate.

The solution is to have each processing element performs a fixed number of iterations and use a flag register to represent whether it is still in the loop. This flag is used to synchronize all processing element to terminate.

In compact Genetic Algorithm (cGA) ([5]), a probability vector is used to represent the population. The length of the vector is equal to the problem size. Rather than trying to represent all of the population in one processing element that requires a large resource. Fig.8 shows pseudo code of cGA. Each processing element is responsible for each gene of chromosome. That is general purpose registers in each processing element represents i-th position of probability vector, an individual a, an individual b, and also other necessary memory for the calculation. In the experiment, cGA is used to solve the one-max problem, a simple problem consisting in maximizing the number of ones of a bitstring.

```
For each pixel on the screen do:
{
  x0 = scaled x co-ordinate of pixel in the
interval (-2.5,1)
  y0 = scaled y co-ordinate of pixel in the
interval (-1,1)

  x = 0
  y = 0

  iteration = 0
  max_iteration = 1000

  while ( x*x + y*y < (2*2)  AND  iteration <
max_iteration )
  {
    xtemp = x*x - y*y + x0
    y = 2*x*y + y0
    x = xtemp
    iteration = iteration + 1
  }

  if ( x*x + y*y < (2*2) )
  then
    color = black
  else
    color = iteration

  plot(x0,y0,color)
}
```

Figure 7. Pseudo code of the Mandelbrot set

```
Compact GA parameters:
n: population size.
l: chromosome length.

for i = 1 to l do
   p[i] = 0.5;
repeat
   for i = 1 to l do
   a[i] = { 1 with probability p[i]
          { 0 otherwise
   b[i] = { 1 with probability p[i]
          { 0 otherwise
   endfor
   // Fitness calculation
   f_a = fitness(a)
   f_b = fitness(b)
   for i = 1 to l do
      if f_a ≥ f_b then
         if a[i] = 1 and b[i] = 0 then
            p[i] = min(1, p[i] + 1/n)
         if a[i] = 0 and b[i] = 1 then
            p[i] = max(0, p[i] - 1/n)
      else
         if a[i] = 1 and b[i] = 0 then
            p[i] = max(0, p[i] - 1/n)
         if a[i] = 0 and b[i] = 1 then
            p[i] = min(1, p[i] + 1/n)
      endif
   endfor
until each p[i] ϵ {0,1}
```

Figure 8. Pseudo code of compact Genetic Algorithm

Table II shows the report on running three programs on the proposed processor, and Table III shows the report on running three programs on the proposed processor in comparison, in term of "number of clock cycle" used to execute the programs (We use number of clock cycle instead of execution time because of tremendous difference between clock frequency of FPGA board and commercial CPU). The programs are written in C programming language and compiled by gnu C compiler, on Intel Core2 Duo CPU P8400 @2.26 GHz. By normalizing the clock frequency, the proposed processor with four processing elements can outperform C program with Intel CPU. In case of cGA, one possible cause is the different method of generating random number, where the hardware method has the advantage. It suggests that the proposed processor can be comparable with Intel CPU for some tasks, and has potential to be improved.

TABLE II. REPORT ON RUNNING THREE PROGRAMS ON THE PROPOSED PROCESSOR

Program	Line of code*	Static Program size (words)	No. of cycle executed
4x4 matrix multiplication	82	172	1,400
Mandelbrot Set (64x64 pixel)	59	80	39,264,548
cGA (256 population)	48	42	244,986

* In Assembly Language

The design is realized on a FPGA. Using the Xilinx devices and their synthesizer software, Design Suite, the total equivalent gate count is 236,071. The maximum frequency is 117 MHz. The general purpose registers consume more than half of the resource. This indicates the area for future optimization. The synthesis result with four processing is given in Fig.9.

```
Target information:
    Vendor:     Xilinx
    Family:     Spartan3
    Device:     XC3S1500L
    Speed: -4

Design Summary:
    Number of Slices:       4,661 out of 26,624 17%
        Slice Flip Flops:       4,525
        4 input LUTs:       12,625
    Number of bonded IOBs:  57 out of 221 25%
    Number of GCLKs:        2 out of 8 25%
Total equivalent gate count for design (w/o block ram): 236,071
Additional JTAG gate count for IOBs:    2,736

Design statistics:
    Minimum period:         8.522 ns
    Maximum frequency:          117 MHz
```

Figure 9. Synthesis result for four processing elements.

The total equivalent gates needed in Micro Blaze system, the processor and 8kbytes of block ram is 315,528 gates [6]. The Micro Blaze alone needs about 55,000 gates. In comparison, the proposed design with 1kbytes of block ram (enough for each of the three programs) needs 268,637 gates. In term of performance, Micro Blaze's maximum frequency is 91 MHz while the proposed design is 117 MHz.

In the aspect of scalability, Fig.10 shows the relevance of total equivalent gate count for design (without block ram) and the number of processing elements in the proposed processor. The more processing elements does the processor contain, the higher potential of speed up of computation it can give.

TABLE III. THE COMPARISON OF NO. OF CYCLE EXECUTED BETWEEN THE PROPOSED PROCESSOR AND C PROGRAM

Problem	C	Proposed Design	Comparison
4x4 matrix multiplication	2,434	1,400	58%
Mandelbrot Set (64x64 pixel)	66,896,000	39,264,548	59%
cGA (256 population)	635,060	244,986	39%

* Executed in integer format

TABLE IV. THE COMPARISON BETWEEN THE PROPOSED PROCESSOR AND XILINX MICRO BLAZE [6]

Category	Xilinx Micro Blaze	Proposed Design	Comparison
Circuit Size (Equivalent Gate)	315,528*	268,637**	85.1%
Maximum frequency (MHz)	91	117	128.6%

* 8kbyte of Block ram is included in the equivalent gate count of Micro Blaze processor.
** 1kbyte of Block ram is included in the equivalent gate count of proposed design.

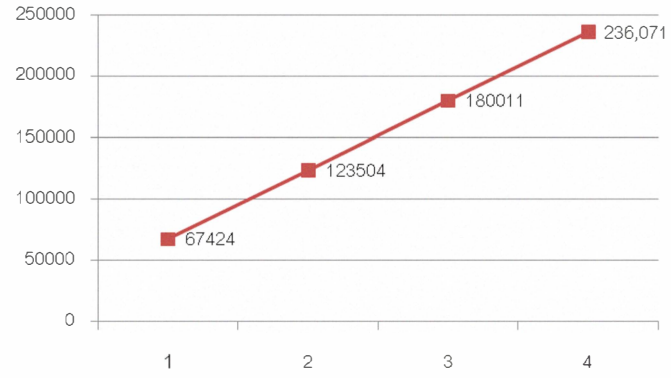

Figure 10. Relevance of total equivalent gate count for design (w/o block ram) and the number of processing elements in the proposed processor

This design is interesting because it has advantages in parallel computing and compactness. Moreover, it is flexibility to scale up by adding more processing elements to attain more speedup. There is room to implement this design as IP-core of embedded system to solve some computation problems, with the need of low power consuming and parallel computing.

V. CONCLUSIONS

This paper presents a design of a softcore of a GPU-styled processor. The design is potentially scalable by adding more processing elements. The performance of the proposed design is satisfactory. The data from the experiment shows a good speedup when compared to test program running on a commercial CPU. This design is suitable for embedding. In fact, a special unit is included in the reference design to demonstrate this flexibility. In the cGA program, a random number generator is realized in hardware. A similar idea can be used to specialize the design to a specific purpose.

REFERENCES

[1] Aporntewan, C.; Chongstitvatana, P.; A hardware implementation of the Compact Genetic Algorithm, Evolutionary Computation, 2001. Proceedings of the 2001 Congress on, vol.1, no., pp.624-629 vol. 1, 2001 doi: 10.1109/CEC.2001.934449.

[2] Thontirawong P.; Burutarchanai A.; Rimcharoen S.; Chongstitvatana P.; Running Compact Genetic Algorithm on Large Scale Problems Using Graphics Processing Unit, In Proceedings of 26th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC 2011), Busan, Korea, June 19-22, 2011.

[3] Advanced Micro Devices, Inc. Evergreen Family Instruction Set Architecture, Instructions and Microcode. Reference Guide. February, 2011.

[4] Xilinx, Inc. MicroBlaze Processor Reference Guide. Available: www.xilinx.com.

[5] Harik, G.R.; Lobo, F.G.; Goldberg, D.E.;, The compact genetic algorithm, Evolutionary Computation, IEEE Transactions on , vol.3, no.4, pp.287-297, Nov 1999 doi: 10.1109/4235.797971.

[6] Satayavibul, C.; Chongstitvatana, P.;, An embedded processor with instruction packing, Electrical Engineering, Electronics, Computer, Telecommunications and Information Technology (ECTI) International Conference, Chiang Rai, Thailand, 9-12 May 2007, pp.1135-1138.