FERMILAB-CONF-07-129-E

Resource Awareness FPGA Design Practices for Reconfigurable Computing: Principles and Examples

Jinyuan Wu

***Abstract*— Computation ability of an FPGA device is determined by three factors: clock frequency, number of logic elements available and efficiency of resource usage, i.e., amount of useful computing works done by unit number of logic elements per clock cycle. The increase of resource is primarily the result of technology progress while the efficient use of the resources is the responsibility of the users. In this document, a variety of examples of the FPGA application in the high-energy physics and accelerator instrumentation will be discussed with emphasis on resource awareness issues. For the FPGA/reconfigurable computing, rich experiences can be transplanted from micro-processor counterpart. While on the other hand FPGA specific issues should be dealt with differently. Several principles in both aspects will be summarized. Topics of this document include: (1) Recognizing FPGA and microcomputer resources, similarities and differences. (2) Flatten designs vs. sequential designs. (3) Principle of loop reduction. (4) Inexplicit computing and hidden resources.**

***Index Terms*— FPGA Firmware, FPGA Computation, Reconfigurable Computing.**

I. Introduction

IN high-energy physics experiment, FPGA devices have been broadly utilized in the trigger and DAQ systems. Examples of the FPGA application are seen in the several areas: (1) glue logic, (2) signal digitization including TDC and ADC, (3) data organization including zero-suppression, event building etc. (4) communication and (5) reconfigurable computing. The total number of useful computations done by a given FPGA device in unit time is a product of clock frequency, amount of logic elements and algorithm efficiency. The electronics technology progress has increased the clock frequency and the number of transistors in a device nearly continuously. On the other hand, the improvement of algorithm efficiency is discouraged under the expectation of continuous technology progress. In practical design work, a bigger, slower and costly scheme can become usable after a faster and larger FPGA (or micro-processor) becomes available in the market. The accumulation of design examples and "successful experience" under this climate will become a problem both in technical and sociological domain.

In this paper, several examples of the FPGA application in the high-energy physics and accelerator instrumentation will be discussed with emphasis on resource awareness issues. Saving resource will not only benefit today's work, but will also benefit our future works when a design is reused.

Manuscript received May 15, 2007. This work was supported in part by Universities Research Association Inc. under Contract No. DE-AC02-76CH03000 with the United States Department of Energy.

J. Wu is with Fermi National Accelerator Laboratory, Batavia, IL 60510 USA (phone: 630-840-8911; fax: 630-840-2950; e-mail: jywu168@fnal.gov).

II. Recognizing FPGA and Microcomputer Resources

We use a simple example shown in Fig. 1 to illustrate similarities and differences of micro-processor and FPGA resources.

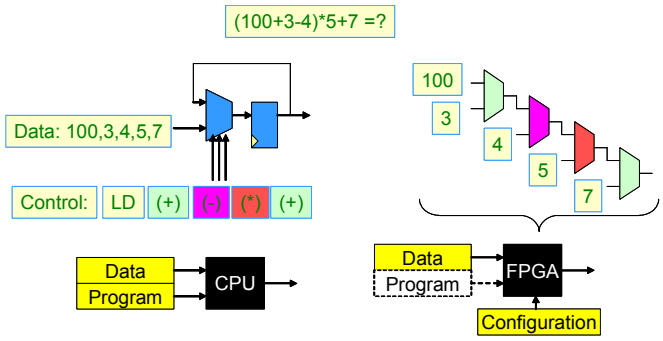

Fig. 1. Comparison of micro-processor and FPGA

Most of today's micro-processor is based on the arithmetic-logic unit (ALU). The ALU as well as other process units change their functionality depending on the control signals which are generated by decoding the micro-instructions. The users provide data to be processed and a sequence of micro-instructions or program to perform the computing tasks.

In FPGA, the computations such as addition, subtraction and multiplication etc. can be done in several process units, rather than in one ALU. The program of the computing tasks is specified not only as a sequence, but also as the interconnection of the process units.

In terms of information flow, micro-instructions of the program are fetched typically once every clock cycle in micro-processor. In FPGA, on the other hand, it is possible to let the program or the interconnections of process units to be stationary while let data to be pipelined through.

It is certainly possible to implement a micro-processor in an FPGA. Sometimes, FPGA is a good prototyping platform for

digital circuits including micro-processors. However, the flexibility of FPGA is at the cost of high transistor usage. Using FPGA to simply duplicate micro-processors is not economical in terms of cost and power consumption. The most attractive feature of FPGA is that the computing architectures in FPGA need not to be the same as in micro-processors. The users are allowed to construct their own architecture suitable for their own computing tasks.

III. Flatten Designs vs. Sequential Designs

The micro-processors are fully sequential designs. Computing works are performed one instruction at a time. In the FPGA, on the other hand, a fully flatten designs can be implemented. In the fully flatten designs, different computation steps are performed by different process blocks.

The advantage of fully flatten design is potentially large data throughput. However, the drawback is the large logic element usage. If in a given computing task, the data throughput is not extremely high, for example, if there are a few clock cycles between two new input data, it is more economical to utilize partially flatten and partially sequential designs as shown in Fig. 2.

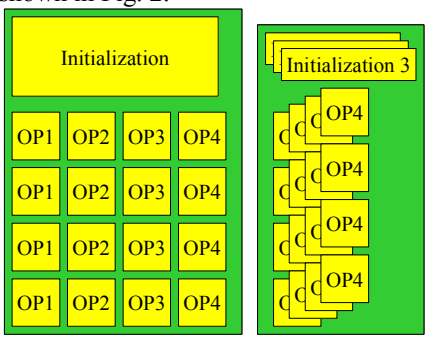

Fig. 2. A fully flatten design and a partially sequential design

Sequencing a process does not reduce number of total computations. It only reuses the silicon resource in several clock cycles so that several computations can be performed in one set of process resource. It is a simple but very useful trick in the FPGA design works.

It should be pointed out that rarely-used processes need sufficient optimizations. This is in contrary with the experiences gained in software programming in which only the most frequently executed program segment needs to be optimized. For example, the system initialization process is only used each time when the system is powering up or is being reset, but it may need large amount of logic elements if designed in flatten fashion. In FPGA, the logic elements dedicated for rarely-used processes cost the same as the ones for the frequently-used processes. So it is important to review resource usages of system initialization or similar processes and consider sequencing the processes if it is necessary.

Sequence control is normally implemented using either finite state machines (FSM) or embedded micro-processor cores. When an input data item is to be fed through a fast and very simple process, typically using a few clock cycles, FSM is a suitable means of sequence control. FSM also responds to external conditions promptly and accurately. However, the sequence or program in the FSM is not easy to change and debug, especially when irregularities exist in the sequence. Embedded microprocessor is another option of sequence control. The drawback of a microprocessor is the large resource usage. The micro-processor is a better choice only if a data item is to be processed with a very complicated program, typically using thousands of clock cycles.

When a data item is to be processed with a medium length program, e.g., using a few hundred clock cycles, a micro-sequencer becomes a better option. We have developed a micro-sequencer called the Enclosed Loop Micro-Sequencer (ELMS) [1] as shown in Fig. 3.

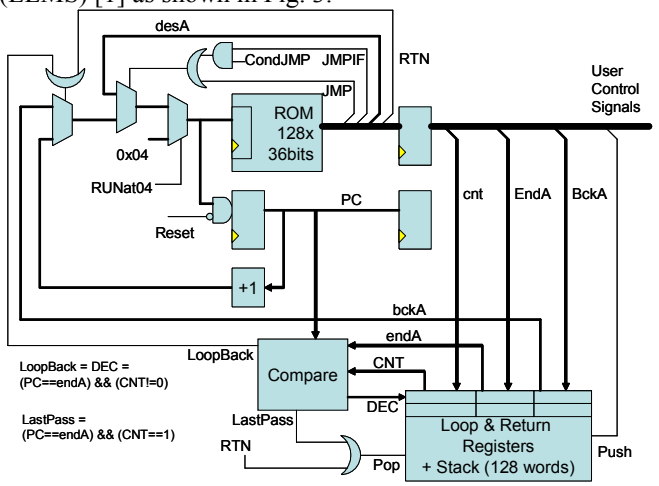

Fig. 3. Detailed block diagram of the Enclosed Loop Micro-Sequencer (ELMS): The Loop & Return Registers + Stack block provides support of the "FOR" loop with constant iterations.

The primary difference between the ELMS and regular micro-processor/micro-sequencer is that the ELMS supports "FOR" loops with predefined iterations at the machine code level and it is self-sufficient to run multi-layer nested-loop programs.

The ELMS shares many micro-structures with typical micro-processors. However, it is not a micro-processor intrinsically since it does not process data. The data are processed by external data process blocks that is sequenced by the ELMS via toggling the user control signals.

IV. The Principle of Loop Reduction

As mentioned earlier, sequencing a process does not reduce number of total computations. It only trades off silicon size with clock cycles. The most fundamental resource saving approach is to reduce total number of computations. The places where most computations can be reduced are inside loops, especially inside the inner-most layer of nested loops, in which each micro-instruction is repeatedly executed multiple times in different iterations. We loosely refer this kind of practices as "loop reduction".

There are two types of loop reduction: (1) to reduce total number of iterations and (2) to reduce number of computations within each pass. There are rich experiences we can borrow from micro-computing in both type of loop reductions. In the

Fast Fourier Transform (FFT), for example, the number of iterations is reduced from $O(n^2)$ in Discrete Fourier Transform (DFT) to O(n*log(n)), where n is number of data points. Multiplier-less algorithms, on the other hand, are examples of reducing computations within the loops.

A few examples will be discussed in the following subsections.

A. The FPGA Track Fitters

Consider a curved track in a multi-plane detector as shown in Fig. 4. Hit coordinates y_i on the planes are measured.

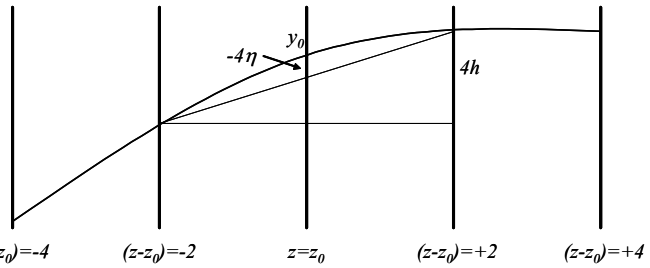

Fig. 4. Curved track in a multi-plane detector

The curved track is modeled by the following equation:

$$y = y_0 + h(z - z_0) + \eta(z - z_0)^2 \quad (1)$$

The track fitting or calculating the parameters in the equation above from the hit coordinates is normally considered a software task. The formulae of calculating the parameters can be written as inner products of hit points and a set of predefined coefficients. It is possible to implement inner products using FPGA as shown in Fig. 5(a).

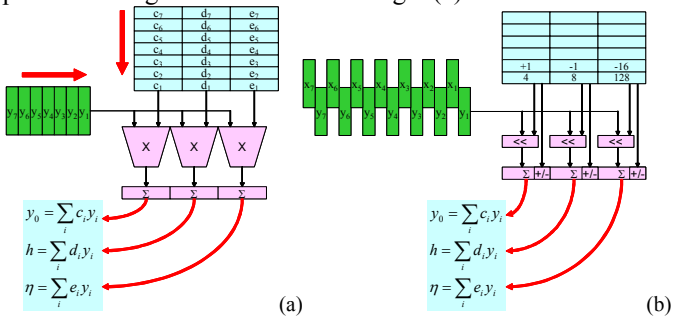

Fig. 5. The FPGA track fitters: (a) regular, (b) multiplier-less approaches.

An example of choosing coefficients for fitting the curvature of the tracks with odd numbers of hits is shown in Table I. The columns of e_i in Table I represent coefficients derived from the least-squares fitting.

The e[i] coefficients are chosen in a spread sheet, guided by the e_i coefficients and they are "two-bit" numbers. This is the basis of multiplier-less implementation of the FPGA track fitter [2] shown in Fig. 5(b).

The multiplication in the inner product calculation is replaced with two shift-add/sub operations. Logarithmic shifters, instead of multipliers are used in the multiplier-less implementation.

Since the parameterization of the track in Equation (1) is chosen with symmetry around z_0, the coefficients for the least-squares fitting are also symmetric. In our work, the coefficient selection is semi-automatic, partially for purpose of our own better understanding to the problem. Clearly it is not too difficult to write a program that chooses the coefficients automatically.

TABLE I
COEFFICIENTS FOR THE FPGA TRACK FITTER (CURVATURE, ODD HITS)

	Half-length of the Track													
	16		14		12		10		8		6		4	
$z-z_0$	e_i	e[i]	e_i	e[i]	e_i	e[i]	e_i	e[i]	e_i	e[i]	e_i	e[i]	e_i	e[i]
-16	5.3	6												
-14	3.3	2	7.5	8										
-12	1.6	2	4.3	4	11.3	12								
-10	0.1	0	1.6	2	5.6	5	17.9	18						
-8	-1.1	0	-0.7	-2	1.0	1	7.2	7	31.0	31				
-6	-2.0	-3	-2.4	-2	-2.6	-4	-1.2	-1	7.8	8	61.0	56		
-4	-2.6	-3	-3.6	-5	-5.1	-5	-7.2	-8	-8.9	-9	0.0	12	146.3	144
-2	-3.0	-3	-4.4	-4	-6.6	-5	-10.7	-9	-18.8	-20	-36.6	-40	-73.1	-64
0	-3.2	-2	-4.6	-2	-7.2	-8	-11.9	-14	-22.2	-20	-48.8	-56	-146.3	-160
2	-3.0	-3	-4.4	-4	-6.6	-5	-10.7	-9	-18.8	-20	-36.6	-40	-73.1	-64
4	-2.6	-3	-3.6	-5	-5.1	-5	-7.2	-8	-8.9	-9	0.0	12	146.3	144
6	-2.0	-3	-2.4	-2	-2.6	-4	-1.2	-1	7.8	8	61.0	56		
8	-1.1	0	-0.7	-2	1.0	1	7.2	7	31.0	31				
10	0.1	0	1.6	2	5.6	5	17.9	18						
12	1.6	2	4.3	4	11.3	12								
14	3.3	2	7.5	8										
16	5.3	6												
Error	2.91	3.02	3.05	3.15	3.22	3.26	3.41	3.43	3.65	3.65	3.93	3.99	4.28	4.29
Ratio		1.04		1.03		1.01		1.00		1.00		1.02		1.00

The relative errors contributed by the parameter η for both algorithms are calculated. The error here is defined as transverse reconstruction RMS error after projecting the track by half-length ($L/2$) from first or last hit of the track, with unit of the RMS error for the y[i] measurements. Assume the errors of y[i] measurements δy_i are independent and they have a same RMS value δy, then the error of calculating parameter η can be estimated:

$$\delta\eta = \delta y \sqrt{\sum_i (e[i])^2} / 4096 \quad (2)$$

It can be seen that the measurement errors for the multiplier-less algorithm are only slightly larger than the one from the mathematically perfect least square algorithm.

B. The Tiny Triplet Finder

To identify and to confirm a straight-line segment in a plane with 2 parameters, for example, at least 3 hits that satisfy a constraint are needed. The 3 hits are grouped together to form a data item called "triplet". Straightforward software implementation of such a function would require $O(n^3)$ execution time, where n is number of hits per plane, in order to examine all possible combinations of three hits using three layers of nested loop. In FPGA hardware implementation, this execution time must be reduced to $O(n)$, to match the time required to fetch the data. The execution time is reduced by "unrolling" two layers of loops, which consumes a significant amount of silicon resources in FPGA devices. The number of logic elements needed in many typical triplet finding implementations is $O(N^2)$ where N is the number of bins that each plane is divided into.

An algorithm, the Tiny Triplet Finder (TTF) [3] was developed for triplet finding. The logic element usage of the TTF implemented in FPGA devices is $O(N \log(N))$ which is significantly smaller than $O(N^2)$ when N is large.

V. The Inexplicit Computing and Hidden Resources

In FPGA, computing can be inexplicitly performed in logic circuits not traditionally considered as computing resources. An example, the time stamp ordering (TSO) module [4] designed for Fermilab BTeV experiment [5] is shown in Fig. 6.

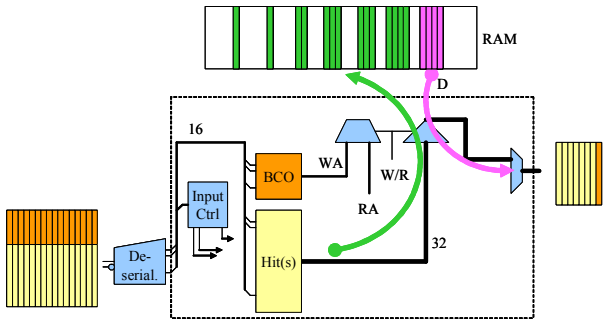

Fig. 6. The time stamp ordering process

The time stamp ordering module receives a cable of 12 optical fibers, 2.5Gb/s each. It stores the random hit data according to the beam cross-over (BCO) number (time stamp) of the hit in temporary memories. After a sufficiently long period of time and the hit data from a BCO are believed have all arrived, the hit data in a BCO are output together to the pixel pre-processor (PP) stage.

To perform the time stamp ordering function, 4 FPGA devices (Altera EP1C6Q240) [6] are use in a TSO module. Data from 3 fibers are handles in one FPGA. Each FPGA is connected to 2 zero bus turnaround (ZBT) synchronous random access memories (SRAM) of size 128K x 32 bits running on a 125 MHz clock. The memories are deep enough to store up to 128 non-empty BCO buckets, which are more than an accelerator turn worth of data. Each FPGA outputs data to 8 differential pairs at a data rate of 375 Mb/s per pair. Each differential pair is routed via the backplane to a PP module. Under normal operation, data from 3 fiber channels in a non-empty BCO are sent to a pre-defined PP module with rotational order.

The data rate of the differential pairs is chosen to be as low as possible to simplify the design of the interconnection and the receivers in the later stages. The reader may note that the total output bandwidth of the TSO module is smaller than the input bandwidth. The TSO operation is a natural lossless compression process since only one BCO number is needed to be attached to a set of detector hits in the output stream.

The time stamp ordering is a computing task inexplicitly performed that yields a reduction of output data rate. The computing is performed essentially by the memory devices. In this design, it is crucial to use true random access memories rather than dynamic memories that only allow limited random access.

VI. Conclusion

Resource awareness not only saves direct cost, but also indirect cost like power consumption, PC board layout, cooling etc. In a lot of time, unnecessary artificial complexities of a system confuse people, often including its own designers.

Necessity of resource saving in FPGA design, or in even broader range, the necessity of resource saving in general, should be viewed in long term. It affects the future of electronics and computing in both technical and sociological aspects.

Code reusing is an almost certain trend in FPGA computing just as in its counterpart of micro-processor computing. Designers should keep in mind that a functional block designed today might be reused thousands times in the future. Today's design could become our library or intellectual property. If the block is designed slightly too big than it should be, it will be too big in thousands of occurrences in the future projects.

What's even worse is that we may gain wrong experiences from these too-big-blocks. The fear that the firmware won't fit causes the planners to reserve excessive costly FPGA resources on printed circuit boards. It is also possible that functions can be mistakenly considered too hard to be implemented in FPGA and resulting in decisions either to degrade the system performances or to increase complexities in system architecture.

References

[1] J. Wu et al., "Readout process & noise elimination firmware for the Fermilab Beam Loss Monitor system", in *Proc. 15th IEEE–NPSS Real Time Conf.,* Batavia, IL, 2007.

[2] J. Wu et al., "FPGA curved track fitter with very low resource usage", IEEE Nuclear Science Symposium Conference Record, p 1290, 2006.

[3] J. Wu et al., "The application of tiny triplet finder (TTF) in BTeV pixel trigger", IEEE Tans. Nuclear Science, vol. 53, p 671, 2006.

[4] J. Wu et al., "Integrated Upstream Parasitic Event Building Architecture for BTeV Level 1 Pixel Trigger System", IEEE Tans. Nuclear Science, vol. 53, p 1039, 2006.

[5] E.E. Gottschalk, BTeV detached vertex trigger, Nucl. Instrum. Meth. A 473 (2001) 167.

[6] Altera Corporation, "Cyclone FPGA Family Data Sheet", (2003) available via: {http://www.altera.com/}