Auto-Configurable Array for GCD Computation
EXTENDED ABSTRACT

Tudor Jebelean
RISC-Linz
A-4232 Hagenberg, Austria
Tel: +43 (7236) 3231-46, Fax: +43 (7236) 3231-30
e-mail: tudor@risc.uni-linz.ac.at

Abstract

A novel one-directional pass-through array for the computation of integer greatest common divisor is designed and implemented on Atmel FPGA. The design is based on the plus-minus GCD algorithm and works in LSB pipelined manner. In contrast with previous designs, the length of the new array is independent of the length of the operands: arbitrary long integers can be processed in multiple passes. The array is auto-configurable: at each step, one new cell is configured according to the input from the previous computation. Preliminary experiments show that for 100 bits a speed-up of 4 over software can be obtained using one 6010 Atmel chip.

Introduction

An emerging trend in scientific computing (e.g. symbolic computation) is the need for increased precision arithmetic - either very long or arbitrary long floats or rationals. Cryptography and data-compression applications also need fast long integer arithmetic.

Reconfigurable computing offers a valuable alternative to software solutions, promising a spectacular increase in speed. Typically, the length of the operands in such computations vary from one application to another, or even during the same computation. Thus, a fixed-size hardware coprocessor will fail to accommodate the operands at a certain moment. In the case of systolic arrays (which is the mostly used design technique for such arithmetic circuits) various methods have been proposed for "packing" several processors in one [2], such that the size of the accommodated operands increases correspondingly. However, these methods lead to a significant increase of the area consumption and also need full reprogramming of the FPGA chip.

We present a method which avoids these inconveniences. Namely, we design a *pass-through* array for the computation of the integer greatest common divisor (GCD). The two operands are processed serially in LSB manner, and after one pass through the array either the GCD is obtained, or the operands are reduced to shorter ones and the process is reiterated. The size of the array is completely independent of the lengths of the input, but, of course, a longer array will result in increased speed.

The array is *auto-configurable*, that is, the functions of the cells are selected upon the values of the first digits which pass through. This is a technique which goes beyond usual *configurable* computing (e. g. configure before computation) and even beyond *re-configurable* (e. g. configure part of the chip during computation): at each step (clock cycle) a new cell is configured according to the result of the previous computation. Current FPGA designs do not allow a direct approach to this computing style, because the configuration cannot be changed using the signals produced inside the chip. We use Atmel FPGA and simulate auto-configurability by multiplexors. Preliminary experiments show that 100-bit operands can be processed in one pass by a 6010 Atmel chip, with a speed-up of 4 over the software solution.

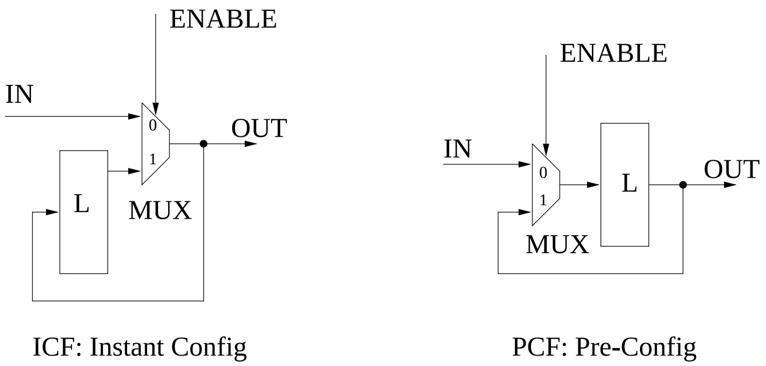

Figure 1: Configuration units.

1 GCD Computation

The computation of the greatest common divisor is the most complicated operation over integers and also the most time consuming in arbitrary precision arithmetic. VLSI implementations are based on the *PlusMinus* algorithm of [1] and on its improvements [4]. We adapt here the later, which consists of two phases:

1. Preprocessing :

- the least-significant common null bits are discarded from both operands A and B;
- after that, if the LSB a_0 of A is null, then the operands are interchanged (after this, a_0 always becomes 1);

2. Reduction is applied iteratively as follows, until B becomes null (we denote by a_0, a_1, b_0, b_1 the LSBs of the operands):

- If $b_0 = 0$ then B is shifted rightward one position ($B \leftarrow B/2$).
- if $b_0 = 1$ then $(A, B) \leftarrow (B, (A \pm B)/4)$, where $\pm$ is *plus* if $a_1 \neq b_1$ and *minus* otherwise.

Correctness and termination of this algorithm are shown in [4].

2 The Array

The *auto-configurability* of the array is done by the configuring units PCF and ICF shown in Fig. 1. PCF (pre-configure) is used when the data which is used for configuration arrives *one clock before* the data which is used for computation (ENABLE signals the later event). ICF (instant configure) is used when the data which is used for configuration arrives simultaneously with the data which is used for computation (ENABLE has to become 1 one clock later).

The **preprocessing** phase of the GCD algorithm is performed by the *feeder* presented in Fig. 2. This unit receives A, B serially, together with the signal START which remains high for the duration of the data transfer. The OR gate detects the first non-zero pair of bits, its output is AND-ed with START in order to trigger the next computation step, which is instant-configured by a_0.

The feeder is followed by a buffer of latches which allows the simultaneous use of the a_0, b_0 and of a_1, b_1 in the subsequent computation.

The **reduction** phase of the algorithm is implemented as an array of identical configurable cells as in Fig. 3. The signal representing b_0 arrives on the line B1 one clock before START becomes high, and it pre-configures the cell to perform either *shift* or *plus-minus*. The shift is performed by delaying A and START through latches.

If the cell is configured for *plus-minus*, then further instant-configuration is done by the XOR of a_1, b_1, which arrive simultaneously with a_0, b_0 when START becomes high. This configuration signal

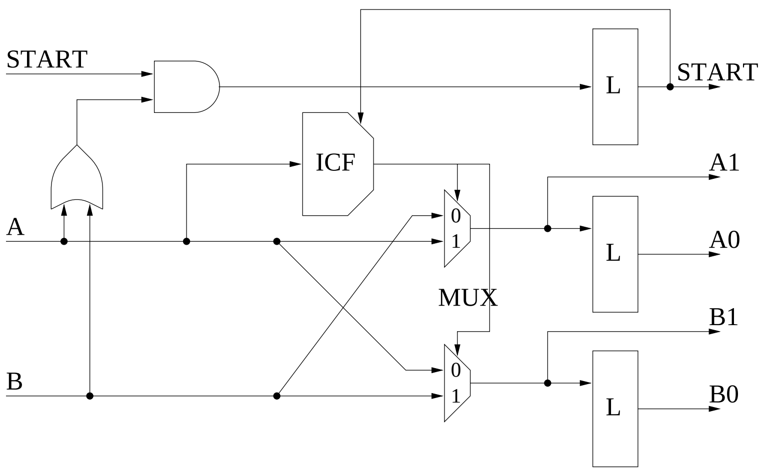

Figure 2: Feeder and subsequent buffer.

commands the arithmetic unit (plus or minus, with carry/borrow feedback incorporated). The result of the arithmetic unit is then supplied as B, while A and START are delayed through two latches – this results in shifting B by two bits.

In order to avoid a high delay through lines B0, B1, one inserts a buffer (as in Fig. 2) after a constant number of cells.

We do not detail here the termination detection and the sign detection, which are solved by a simple *postprocessign* circuit, as well as I/O buffering between the host and the coprocessor and between several chips in a multi-chip design, which is easy to perform due to the unidirectional pipelined style of communication.

Experiments and Conclusions

The array is being implemented using Atmel 6000 FPGA series. We present here a preliminary assesment of the performance.

Each cell uses 22 Atmel logic components, and is simple enough to be manually placed and routed. Due to local communication of the systolic design, several cells are then placed and routed by simple tiling. At 50% utilization this allows the placement of 150 cells on a 6010 chip, including the pre-processing, postprocessing and I/O buffers. This is enough for the processing in one pass of 100-bit operands. (By software experiments we detected that n-bit operands need in average .75 shift steps and .75 plus-minus steps, which gives an average of $1.5n$ cells and $3n$ clock cycles.) The placed circuit runs at 5 Mhz (200 ns clock delay), hence 300 cycles will take 60 μs.

For comparison, GCD computation in a fast C-implementation (see e.g. [3]) on a DEC workstation 5000/200 (about same generation technology as the Atmel FPGA) takes in average 235 μs, hence the speed-up is 4. Note that by implementing a longer array on a multi-chip module the speed-up will increase linearly for longer operands, because the quadratic time-complexity of the seqeuntial GCD algorithm – in contrast with the linear time-complexity of the systolic device.

References

[1] R. P. Brent and H. T. Kung. A systolic algorithm for integer GCD computation. In K. Hwang, editor, *Procs. of the 7th Symp. on Computer Arithmetic*, pages 118–125. IEEE Computer Society, June 1985.

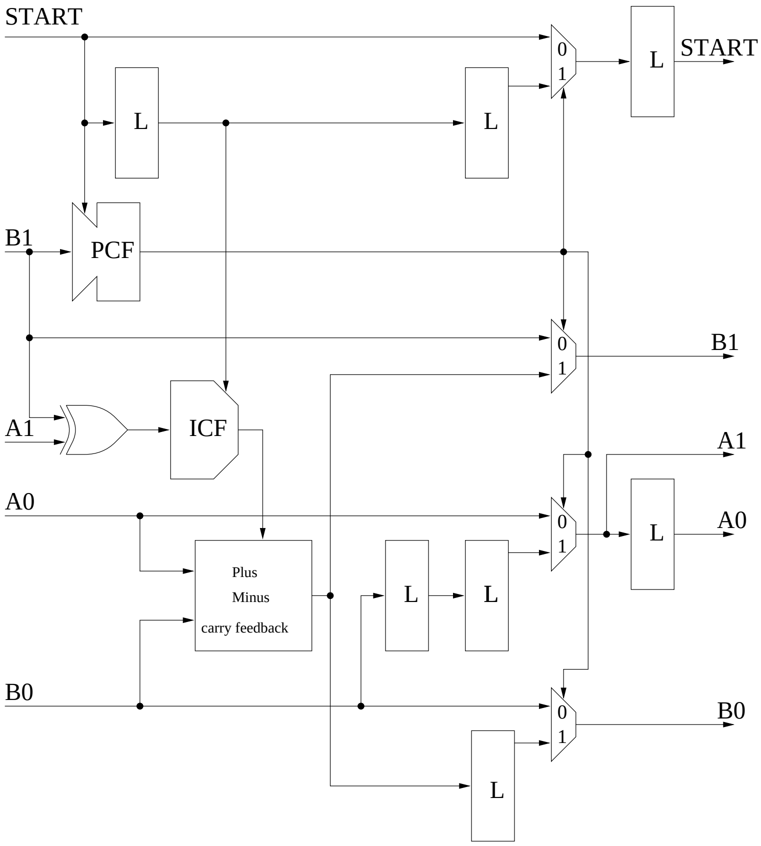

Figure 3: Configurable cell.

[2] P. Dewilde and E. Deprettere. Architectural synthesis of large, nearly regular algorithms. *Ann. Telecom.*, 46(1-2):49 – 59, 1991.

[3] T. Jebelean. Comparing several GCD algorithms. In E. Swartzlander, M. J. Irwin, and G. Jullien, editors, *ARITH-11: IEEE Symposium on Computer Arithmetic*, pages 180–185, Windsor, Canada, June 1993.

[4] T. Jebelean. Design of a systolic coprocessor for rational addition. In P. Capello, C. Mongenet, G-R. Perrin, P. Quinton, and Y. Robert, editors, *ASAP '95, Strasbourg, France, July 1995*, pages 282–289. IEEE Computer Society Press, 1995.