A Fast-Multiplier Generator for FPGAs

Suthikshn Kumar, Kevin Forward and M. Palaniswami
Department of Electrical and Electronic Engineering,
The University of Melbourne,
Parkville, VIC-3052
Australia

Abstract

FPGA implementation of artificial neural networks calls for multipliers of various word length. In this paper, a new algorithm for generating variable word length multipliers for FPGA implementation is presented. The multipliers generated are based on a Booth Encoded optimized Wallace tree architecture. Several features of FPGA architecture are used to generate fast and efficient multipliers. These multipliers are shown to be 20% faster than existing FPGA multiplier implementations.

1 Introduction

Recently, several researchers have implemented artificial neural networks in Field Programmable Gate Arrays(FPGA)[1],[2]. These implementations have achieved very good speeds through the use of finite precision arithmetic. An extensive error analysis of finite precision arithmetic has been carried out by Holt and Hwang[7]. Their results show that the number of bits required to represent different parameters like weights, biases, activation, inputs and outputs vary depending on the algorithm. For example, while 13 to 16 bits are required for representing weights for back propagation learning, 7 to 8 bits are sufficient for forward retrieving. By taking these results into account FPGA implementations can keep the system cost low while achieving the best speeds possible. For implementing finite precision hardware, one needs to design arithmetic units such as adders and multipliers of various word lengths. The recent introduction of X-BLOX design tools[4] is a step in this direction. X-BLOX design tools synthesize efficient adders, comparators, accumulators etc. However X-BLOX design tools do not generate multipliers. In order to speedup the process of multiplier design, XGEN, a tool written in C++, has been developed. XGEN is a multiplier generator and differs from other multiplier generators[9], in that it is particularly developed for FPGA implementation. It takes advantage of XC4000 Xilinx FPGA's internal architecture to generate fast and compact multipliers. Several multipliers have been generated using XGEN and extensive simulations have been carried out to test them. The results show that the multipliers generated are 20% faster than implementations reported in literature[10] for word lengths upto 16-bits. This improved performance is achieved at the cost of higher number of programmable logic gates compared to [10].

The structure of this paper is as follows : In section 2, a brief overview of FPGA architecture is given. In section 3, a brief discussion about Booth encoded Wallace tree multipliers is given. In section 4, the multiplier generator XGEN is described along with the simulation results. In section 5, conclusions of using XGEN are presented.

2 FPGA Architecture

FPGA's store their configuration data in SRAM. Hence these devices can be re-configured to change the logic function while they are embedded in the system. Hardware can be changed as easily as software. XC4000 series is the third generation FPGA's introduced by Xilinx[3]. It supports system clock rates of 40 to 50 MHz. The devices in XC4000 series have programmable logic densities of up to 20,000 gates per chip. These FPGA's comprise of three kinds of configurable basic blocks i.e., configurable logic blocks(CLB's), input/output blocks(IOBs) and Switch matrix interconnections (ICNs). Figure 2.1 shows the block diagram of an FPGA. The CLBs provide functional elements for constructing the user's logic. The IOBs provide the interface between the package pins and internal signal lines. The ICNs provide routing paths to connect the inputs and outputs of the CLBs and IOBs onto the appropriate networks.

1063-9667/95 $4.00 © 1995 IEEE

Customized configuration is established by programming internal RAM cells that determine the logic functions and interconnections implemented in the FPGA.

Each CLB of the XC4000 series FPGA has two 4-input function generators, a 3-input function generator and a pair of flip flops. The function generators can implement arbitrarily defined Boolean functions of their four inputs. Each CLB also includes high speed carry logic that can be activated by configuration. The two 4-input function generators can be configured as a 2-bit adder with built in hidden carry that can be expanded to any length. The fast-carry logic makes the XC4000 FPGA's suitable for implementing high speed arithmetic units.

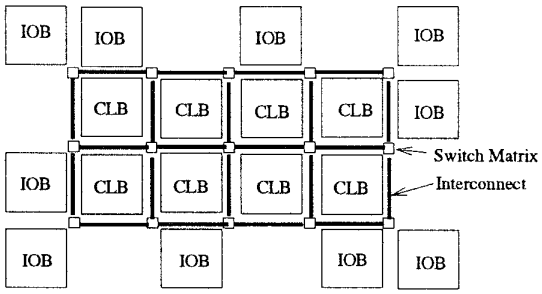

Figure 2.1 FPGA Internal Architecture

3 Multiplier Architecture

Booth, proposed a method for multiplying two signed 2's complement numbers[5]. His method was to use radix 2 recoding of the multiplier. In order to improve the speed of the multipliers higher radix recoding have been proposed[6]. The block diagram of a Booth Encoded Wallace Tree multiplier is shown in figure 3.1. In the current work radix 4 recoding(Modified Booth encoding) of the multiplier has been used. Modified Booth encoding ensures that, only N/2 partial products are generated for an N-bit by M-bit multiplier. These N/2 partial products are added at the Wallace tree.

The multiplier X and multiplicand Y are N and M bits wide respectively. There are N/2 3-bit Booth encoders each taking 3 bits of X and generating control signals. The number of bits inspected is 3 and each cycle will eliminate 2 bits. However there is a flaw in this algorithm requiring a divide-by-2 correction cycle for even number bit length multipliers[8].

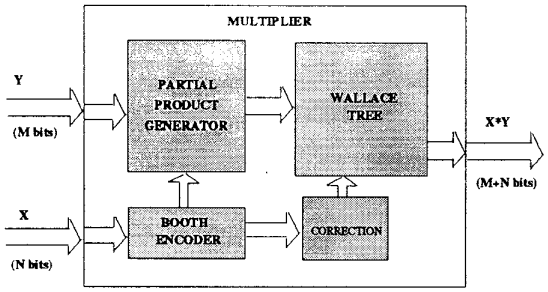

Figure 3.1 Multiplier Architecture

Ardekani[9] has proposed a method for sign extension prevention. When adding partial products, P_i's, the P_{i+1}th partial product is placed two bits to the left of the P_ith partial product. All P_i's should be sign extended to the MN binary position. In the case of Wallace tree adders, since the partial products are added in parallel, sign extension becomes very costly.

We use the multiplier bits directly to generate partial products. This is carried out by using 5-input lookup table which can be implemented in one CLB. Each of these look-up tables will take in 3 inputs from multiplier X and 2 inputs from multiplicand Y and generate one bit of a partial product. To take the 2's complement of the multiplicand Y, a 1 is added to its complement. This is done in the Wallace tree. Instead of adding final carry bits and sum bits of the Wallace tree using a carry select adder, we used a simple ripple carry adder implemented in the FPGA by using dedicated carry logic. The dedicated carry adders are more suited to FPGA architecture. The main reason for using the Wallace tree for partial product summation is speed advantage which it provides. Another advantage is that, the multiplier can be easily converted into a multiplier accumulator.

4 XGEN: Multiplier Generator

XGEN is a C++ program written to generate fast and compact multipliers for FPGA implementation. XGEN is a hierarchical netlist generator. It generates a netlist for the Cadence CAD schematic capture tool, Concept. The different signals and logic blocks are assigned appropriate property values in order to utilize the FPGA specific resources. XGEN uses the LCA4K library, the X-BLOX library and some basic blocks created by us. It is organized into several functions which generate netlists for each block. These are

a Booth encoder, a Partial Product generator, a Wallace tree, a correction block and and hence the overall multiplier. The netlist generation is fully automated. The input to XGEN is the number of bits, N and M.

As the program generates a hierarchical netlist, different components can be independently tested and used. The generated netlist file is converted to XNF format by using proprietary Xilinx software tools. An LCA file for simulation of the multiplier is obtained by using the Xilinx placement, partition and routing tools and the X-BLOX synthesis tools. Cadence's Rapid-Sim tool has been used for simulation of the multiplier. In [9], an algorithm for generating an optimized Wallace-tree has been proposed. To make use of the dedicated carry logic, an improved algorithm is used in the present work. The dedicated carry signal cannot be switched to other interconnects without introducing considerable delay. Also, these signals have been routed among adjacent CLBs along columns of CLBs except at the edges of FPGA where the connections run along the chip edge. Hence the dedicated carry signals should be treated differently from other bits. In the following algorithm, the delays of final sum bits from the Wallace-tree are optimized by connecting the incoming partial product bits and intermediate sum bits in the following way. The bits which arrive at the Wallace tree early, are added first. These bits are determined by sorting the list of bits according to their delays. The X-BLOX tool is used for synthesizing adders.

```
/* LP_i list = list of all partial product bits to ith column of Wallace-tree */
/* N_pi= Number of bits input to ith column of Wallace-tree */
j = 1;
while ( j > 0) {
  j = 0; k = 0;
  for( i = 0; i ≤ (N + M − 2); i++ ) {
    k++;
    switch(N_pi ) {
      case '0' : break ;
      case '1' : if(j == 0){ label the bit of LP_i
                   as output ;
                   remove this bit from LP_i ; }
                   break;
      default : sort LP_i according to delays of
                   each bit ;
                   label top two bits from LP_i as inputs to
                   adder ;
                   remove these two bits from LP_i ;
                   append sum bit to LP_i after computing
                   its delay ;
                   j++ ; m = k;
                   break ;
    }
  }
  synthesize j bit adder using X-BLOX tools ;
  append carry-out bit to LP_{m+1} list ;
}
```

Initially the delays of all the partial product bits entering the Wallace tree are assumed to be equal and the netlist is generated. The implementation is then simulated to obtain the delays of all the partial product bits. These delays are then fed to XGEN to generate the final netlist of the multiplier implementation. This netlist for the Wallace-tree is thus optimized.

The delay of an N-bit adder which uses dedicated carry logic, is given by[3] to be $(8.5 + 0.75N)ns$. This formula is valid for devices with speed grade -5. Since the addition of the partial products take place in parallel the delay of the Wallace tree is proportional to $\lceil \log_2 N \rceil$. Hence, the delay of the Wallace tree portion of the multiplier is approximately $(\lceil \log_2 N \rceil (8.5 + 0.75(M + N)))ns$. Added to this figure is the delay of generating the Partial Products, Booth encoding and interconnect delays. The 5-input function generator is used for generating one bit of Partial Product. The delay of 5-input function generator is 5ns[3]. Assuming 5ns interconnect delay, the delay of the N-bit by M-bit multiplier generated by XGEN is approximately given by

$$t_{md} \simeq (10 + \lceil \log_2 N \rceil (8.5 + 0.75(M + N)))ns.$$

Table 4.1

Multiplier 8-bit	Util (%)	Delay(ns)	
		Calc	Measured
XGEN+XBLOX	20	71.5	80.3
XGEN	23		120.7
Lou94	17		102

An 8-bit by 8-bit multiplier was generated using XGEN and its delay was measured for XC4010PG191-5 implementation by using the Xdelay program which is a part of the Xilinx development tools. The multiplier speeds are shown in table 4.1. The multiplier generated by XGEN is 20% faster than the multiplier

implementation reported in literature[10]. It is observed that upto 16 bits, the new multipliers outperform the multiplier implementation reported in [10], which uses a multiplier architecture called Linear Sequential Arrays(LSA). This is achieved at the cost of increased number of CLBs as indicated by the utilization factor.

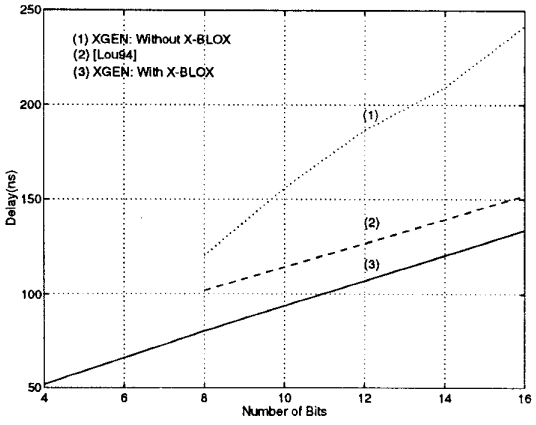

Figure 4.1 Speed performance of Multipliers

XGEN was tested by using it to generate a number of multipliers. The most important attribute of a multiplier is its speed performance. The delay of each multiplier generated was measured using the Xdelay program. The figure 4.1 shows the variation of delay of the multiplier generated as the number of bits is varied(XC4010PG191-5). Another important attribute of the parallel multiplier is the number of CLBs it occupies in the FPGA. The FPGA utilization is defined as the percentage of CLBs used totally or partially to the total number of CLBs present in the FPGA. From the results, utilization for an M by N multiplier is found to be $(0.364(M \times N) - 1.4)\%$. The delay of N-bit by N-bit multiplier is found to be $(7.25N + 2)$ns.

5 Summary and Conclusions

A multiplier generator XGEN has been developed which is able to generate parallel multiplier netlists. The multipliers give the (M+N) bit product of N bit and M bit inputs. The values, M and N are selected by the user. M can be any integer while N should be an even number. By taking advantage of XC4000 series FPGA's internal architecture and the efficient submodules provided in the Xilinx library, XGEN produces multipliers which are not only fast but also utilize relatively small number of CLBs. The multipliers generated by XGEN are 20% faster than the multiplier implementations reported in literature. This is achieved by using a parallel multiplier architecture which requires more CLBs than a sequential multiplier. These multipliers are useful in finite precision hardware implementation of artificial neural networks, digital signal processors and other similar applications.

References

[1] C. E. Cox and W. E. Blanz, " GANGLION- A Fast Field-Programmable Gate Array Implementation of a connectionist Classifier ", *IEEE Journal of Solid State Circuits*, Vol. 27, No. 3, March 1992, pp. 288-299.

[2] J.G.Eldredge and B.L.Hutchings, "Density Enhancement of a Neural Network Using FPGAs and Run-Time Reconfiguration", *Proc. IEEE workshop on FPGAs for Custom Computing Machines*, California, 1994.

[3] The XC4000 Data Book, Xilinx, Inc., San Jose, Calif., 1992.

[4] XACT Macro Libraries, Vol 2: X-BLOX Design tool, Ver 4.1, Dec 1992, Cadence Design Systems, Inc., San Jose, Calif.

[5] A. D. Booth, " A Signed binary multiplication technique ", *Quart. J. Mech. Appl. Math.* , Vol. 4, part 2, pp. 236-240, 1951.

[6] O. l. Macsorley, " High Speed arithmetic in binary computers", *Proc. IRE*, vol. 49, Jan. 1961.

[7] J. L. Holt and J. N. Hwang, " Finite Precision Error Analysis of Neural Network Hardware Implementations", *IEEE Trans on Computers*, Vol. 42, No. 3, March 1993, pp. 281-290.

[8] P. E. Madrid et. al., " Modified Booth Algorithm for High Radix Fixed-Point Multiplication", *IEEE Trans on VLSI Systems*, Vol. 1, No. 2, June 1993, pp. 164-167.

[9] J. F. Ardekani, " M x N Booth Encoded Multiplier Generator Using Optimized Wallace Trees", *IEEE Trans on VLSI Systems*, Vol. 1, No. 2, June 1993, pp. 120-125.

[10] M.E. Louie and M.D. Ercegovac, "A Variable Precision Multiplier for Field Programmable Gate Arrays", *Proc.ACM Second International Workshop on FPGAs*, Feb 1994, Berkeley, CA.