High Performance Multipliers in QuickLogic FPGAs

Introduction

Performing a hardware multiply is necessary in any system that contains Digital Signal Processing (DSP) functionality such as filtering, modulation, or video processing. Often there is an off-the-shelf component that the engineer can use to solve his problem. However, in some designs, the expense of a dedicated DSP chip is not justified and so the use of a Field Programmable Gate Array (FPGA) is more viable. Another reason for using a programmable device to perform hardware multiplication is if the system parameters are non-standard and would require expensive circuitry outside the DSP, or running over-powered DSP chips for a large and small operand combinations (a 12- by 4-bit multiplier, for example). Using an FPGA in cases like these gives complete flexibility, high performance, and lower cost over an off-the-shelf solution.

This application note describes various techniques available to the designer for performing high-speed signed multiply operations in QuickLogic FPGAs. A brief discussion on the basics of digital multiplication will be introduced, followed by discussions on how to implement the signed multiplier to meet your design requirements. Example implementations will be shown, from the slower, purely combinatorial version to the fully pipe-lined version that runs at 200MHz. The example designs shown in this application note are done in schematic as well as in Verilog and VHDL. These are generic designs that can be easily modified to suit various design needs. These are not "hard macros". Therefore they can be modified by users. All timing information presented in this application note is derived from worst case simulations performed within Quick*Works*™, QuickLogic's integrated Windows-based design software.

Two's Complement

Before starting with signed multiplication, a quick review of the 2's complement system of signed number representation for a binary number would be helpful in understanding the derivation of the algorithm. Basically, in the 2's complement system, the left most bit (MSB) indicates the sign of the number, with 0 being positive, 1 being negative. To obtain a negative number, simply subtract the corresponding positive number from 2^n, where *n* is the number of bits of the original number. For example, to obtain the 4 bit signed number `-4`, take 2^4`(b'10000),` and subtract from it the corresponding positive number, `4(b'0100),` the result is (`b'1100`), `-4` in the 2's complement representation.
Alternatively, it can be obtained by subtracting $2^{(n-1)}$ from the corresponding positive number to obtain the negative number. Using the same example, to obtain the number `-4`, take the number `4(b'0100)` and subtract $2^{(4-1)}$`(b'1000)` from it. The result is `(b'1100)`.
The multiplier algorithm discussed in this application note takes advantage of the latter method. One important note on using 2's complement number representation is the need for sign extension. That is, to obtain a negative number using a greater number of bits, simply repeat the sign bit to the left until the desired number of bits are filled. For example, to extend the number `-4 (b'1100)` from 4 bits to 8 bits, the resultant sign extended number would be `(b'11111100).`

Binary Multiply Basics

Similar to the familiar long hand decimal multiplication, binary multiplication involves the addition of shifted versions of the multiplicand based on the value and position of each of the multiplier bits. As a matter of fact, it's much simpler to perform binary multiplication than decimal multiplication. The value of each digit of a binary number can only be 0 or 1, thus, depending on the value of the multiplier bit, the partial products can only be a copy of the multiplicand, or 0. In digital logic, this is simply an AND function. Figure 1 shows the multiplication of two unsigned 4 bit numbers as an example.

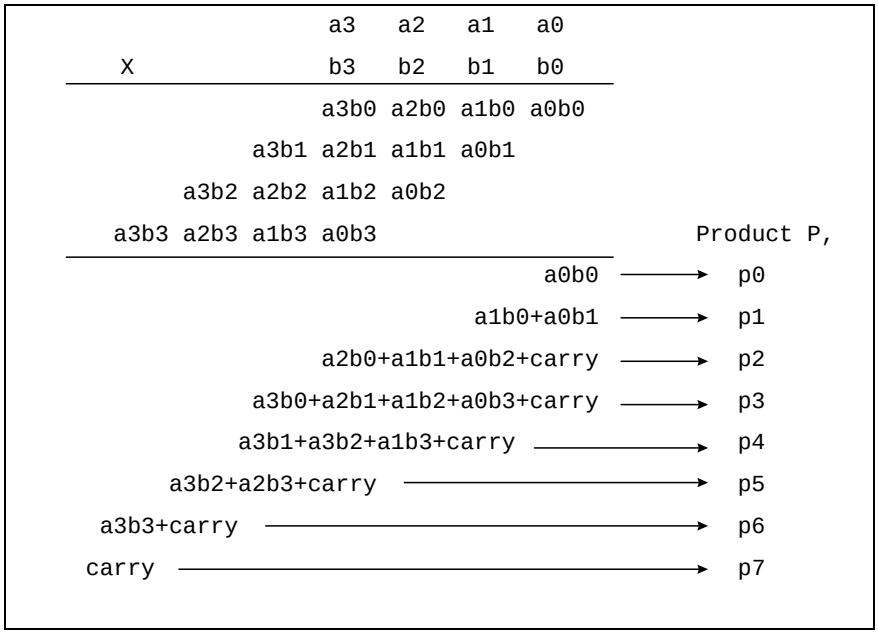

Figure 1. Basic Binary Multiplication

Baugh-Wooley Algorithm

The Baugh-Wooley algorithm for the unsigned binary multiplication is based on the concept shown in Figure 1. The algorithm specifies that all possible AND terms are created first, and then sent through an array of half-adders and full-adders with the carry-outs chained to the next most significant bit at each level of addition. For signed multiplication (by utilizing the properties of the two's complement system) the Baugh-Wooley algorithm can implement signed multiplication in almost the same way as the unsigned multiplication shown above. The algorithm for the signed multiplication is listed in Figure 2.

$$P = a[m-1]b[n-1]2^{(m+n-2)} + \sum_{i=0}^{i=n-2} NOT(a[m-1]b[i])2^{(i+m-1)}$$

$$+ \sum_{j=0}^{j=m-2} NOT(a[j]b[n-1])2^{(j+n-1)} + \sum_{i=0,j=0}^{i=m-2,j=n-2} a[i]b[j]2^{(i+j)}$$

$$+ 2^{(n-1)} + 2^{(m-1)} - 2^{(m+n-1)}$$

Figure 2. Baugh-Wooley algorithm for signed multiplication

Although the equation in Figure 2 looks complicated, the hardware implementation is actually very similar to the unsigned implementation. The difference is that except for the AND term involving the MSB of both the multiplicand and multiplier, all other AND terms involving the MSBs of the multiplicand or the multiplier are inverted before being feed into the adder array. The last two addition terms in the equation are easy to implement, involving only the addition of a 1 to the appropriate adder stage. The last term in the equation is a subtraction, which can be implemented with a XOR gate in the last stage. The architecture block diagram in Figure 3 illustrates the flow for the multiplication of two 4-bit signed numbers using the equation in Figure 2 ($m = n = 4$). The mathematics involved in deriving the equation in Figure 2 are left as an exercise for the reader.

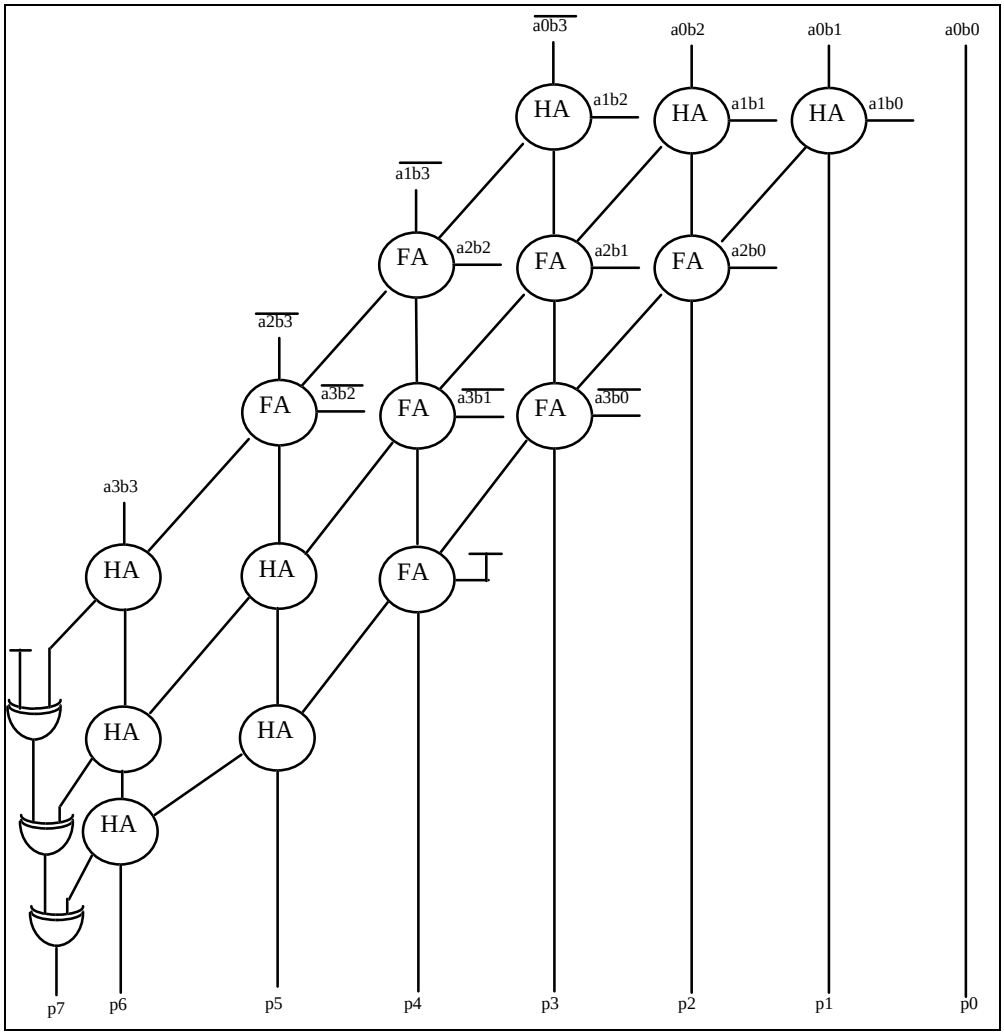

Figure 3: Architectural block diagram of the Baugh-Wooley multiply algorithm.

Implementing an 8-bit signed Baugh-Wooley multiplier

To implement an 8-bit signed Baugh-Wooley multiplier, set $m=n=8$ for the equation in Figure 2. Solve the equation to obtain the AND terms and the arrangement of the adder array. The array approach of the algorithm allows clean hardware implementation that clearly shows the data flow. The schematic in Figure 4 shows the combinatorial implementation of the 8-bit multiplier. Since QuickLogic's pASIC2 and pASIC 3 families of FPGAs can implement a full adder in a single logic cell, the combinatorial 8-bit signed multiplier takes up less than 110 logic cells, less than

20% of the logic resources available in the QL2009 device targeted. This implementation of the multiplier can run at 25MHz.

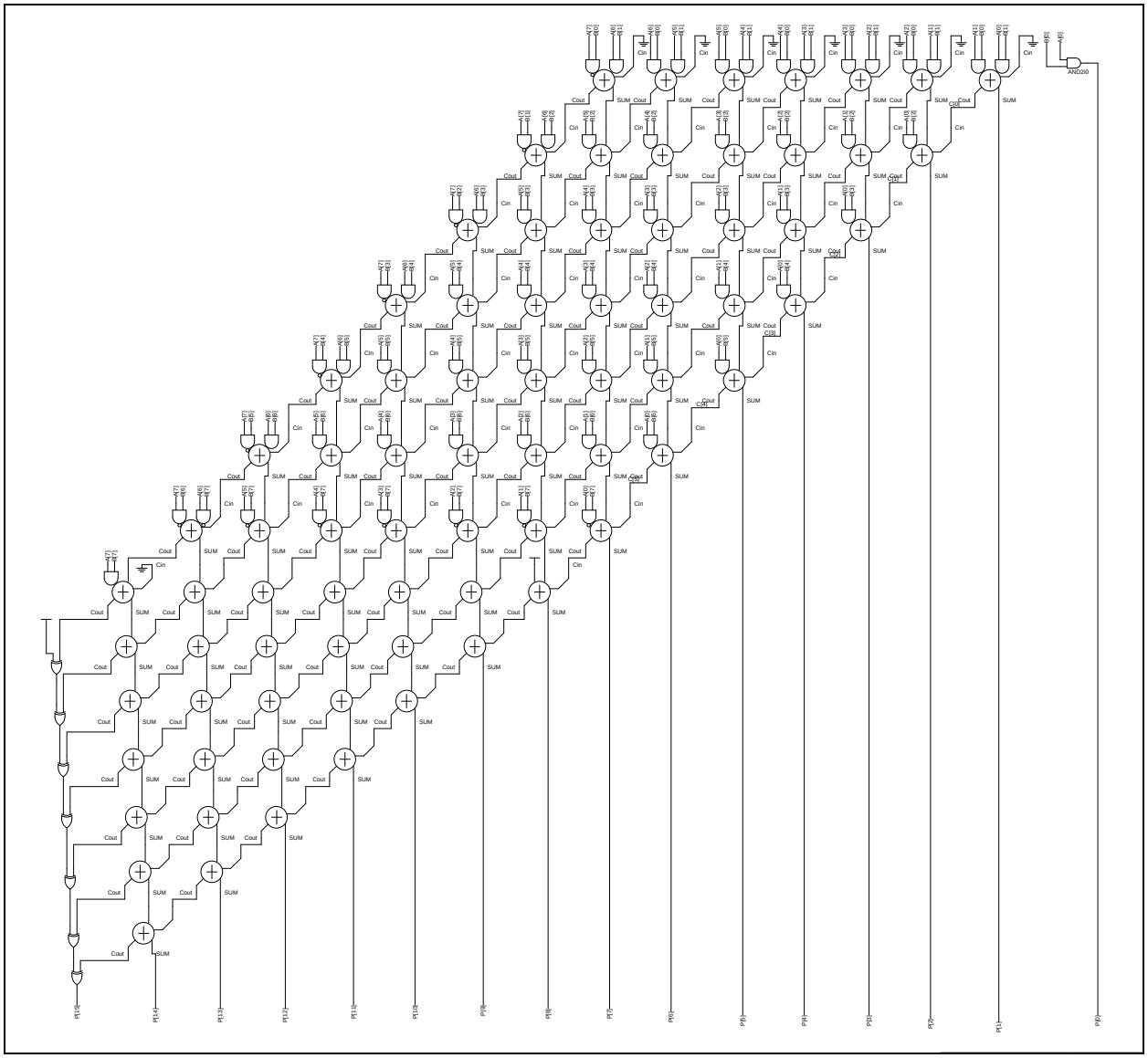

Figure 4: Combinatorial 8-bit Signed Multiplier

In addition to the schematic multiplier illustrated in Figure 4, the synthesis-friendly logic cells in the pASIC2 and pASIC3 families of FPGAs are also well suited for high performance Verilog and VHDL implementations. These HDL versions of the multiplier have comparable performance to the schematic version, while offering the ease of use of HDLs, with powerful features such as scaleable width for design reusability.

To verify the performance of the multiplier, the Path Analyzer static timing analysis tool included with Quick*Works* and Quick*Tools* were used to obtain the flip-flop-to-flip-flop delay (in order to get the true performance of the multiplier, the inputs and outputs were registered). The Silos III Verilog simulator included in Quick*Works* was used to perform both pre-layout functional simulation and post-layout timing simulation. A self-checking Verilog test fixture is used to randomly generate up to 400 input vectors, and automatically notify the user when a failure occurs. A VHDL version of the test fixture is also included for users who wish to use their own VHDL simulator to perform the simulation.

The results for different versions of the combinatorial 8-bit signed multiplier are listed in the ***Comparison of Methods Presented*** section later in the application note.

Speeding Up the Multiplier

Looking at the schematic in Figure 4, it is obvious that the performance of the multiplier can be increased by reducing the carry delay through the half-adder array. Upon closer inspection, one will see that the half-adder array is nothing more than a ripple adder. Since the half-adder array consists of 8 stages, think of it as an 8 bit ripple adder. By replacing the ripple adder with, for example, a carry-select adder, several nanoseconds can be saved. However, although several fast adder options exists, the increase in performance by replacing the ripple adder is limited to no more than 15% due to the delays through the AND/ADD array. To gain higher performance, the multiplier needs to be pipelined.

Pipelining for Performance

Throughput is often a greater concern than one-cycle response in DSP designs. In this case, the engineer may opt to tolerate some latency in the multiply operation in return for a faster maximum clock rate. This is accomplished in a multiplier by breaking the carry-chains up and inserting flip-flops at strategic locations. Care must be taken to ensure that all inputs to an adder are created from signals at the same stage in the pipeline (i.e., the registers must effect a clean break across the matrix of adders). Once the maximum acceptable latency is determined, the placement of the registers needs to be carefully considered. The delay should be evenly distributed across the pipeline, so each stage has the same delay as the other stages. The path delays can be analyzed with the Path Analyzer in Quick*Works* or Quick*Tools*. The locations to insert the pipeline registers can then be determined by the distribution of the path delays. The example in Figure 5 shows the 8-bit signed multiplier in Figure 4 with a single pipeline stage inserted.

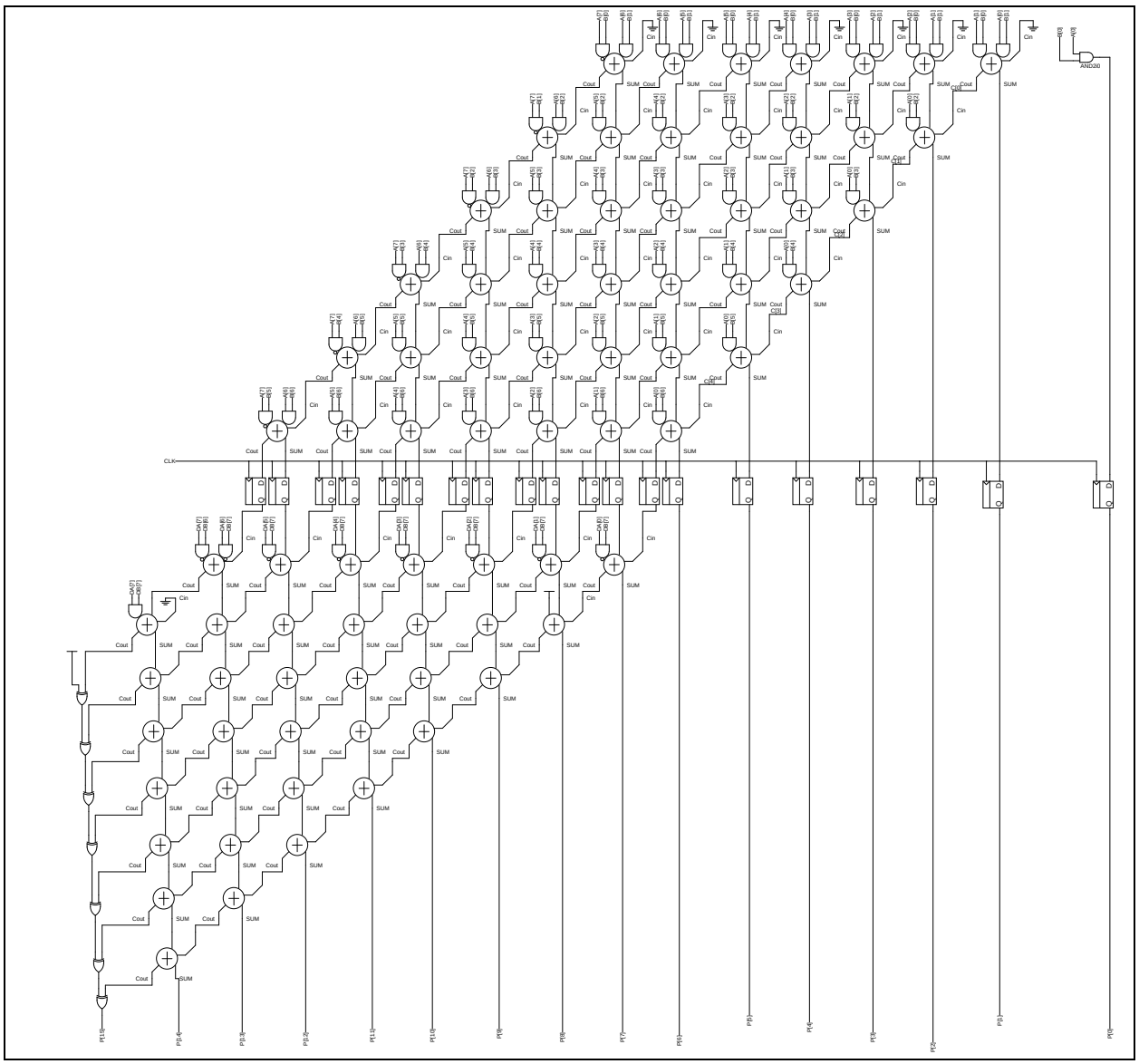

Figure 5: Single Stage Pipelined 8-Bit Signed Multiplier

With just a single stage of pipeline inserted, the multiplier now runs in excess of 35MHz. The Verilog and VHDL test fixtures for the pipelined version of the multiplier are very similar to the test fixtures for the combinatorial version, except the addition of pipeline registers to the expected result that are used to check against the output.

Fully Pipelined 200MHz Multiplier

If the latency is acceptable, the multiplier can be fully pipelined by adding registers between every adder . This would give a latency of 2n-1 clock cycles, where *n* is the number of bits of the inputs. The schematic in Figure 6 shows a fully pipe-lined version of the multiplier that can run up to 200MHz. Notice that in order to control the fan out to no more than 4, a fan out register array is used. This design utilizes 390 logic cells, taking up approximately 75% of the QL2007 device targeted. In order to obtain the 200MHz performance, strict control of the design is needed, therefore only the schematic version of the design can run at 200MHz. The HDL versions of the fully pipelined design run at slower clock rate.

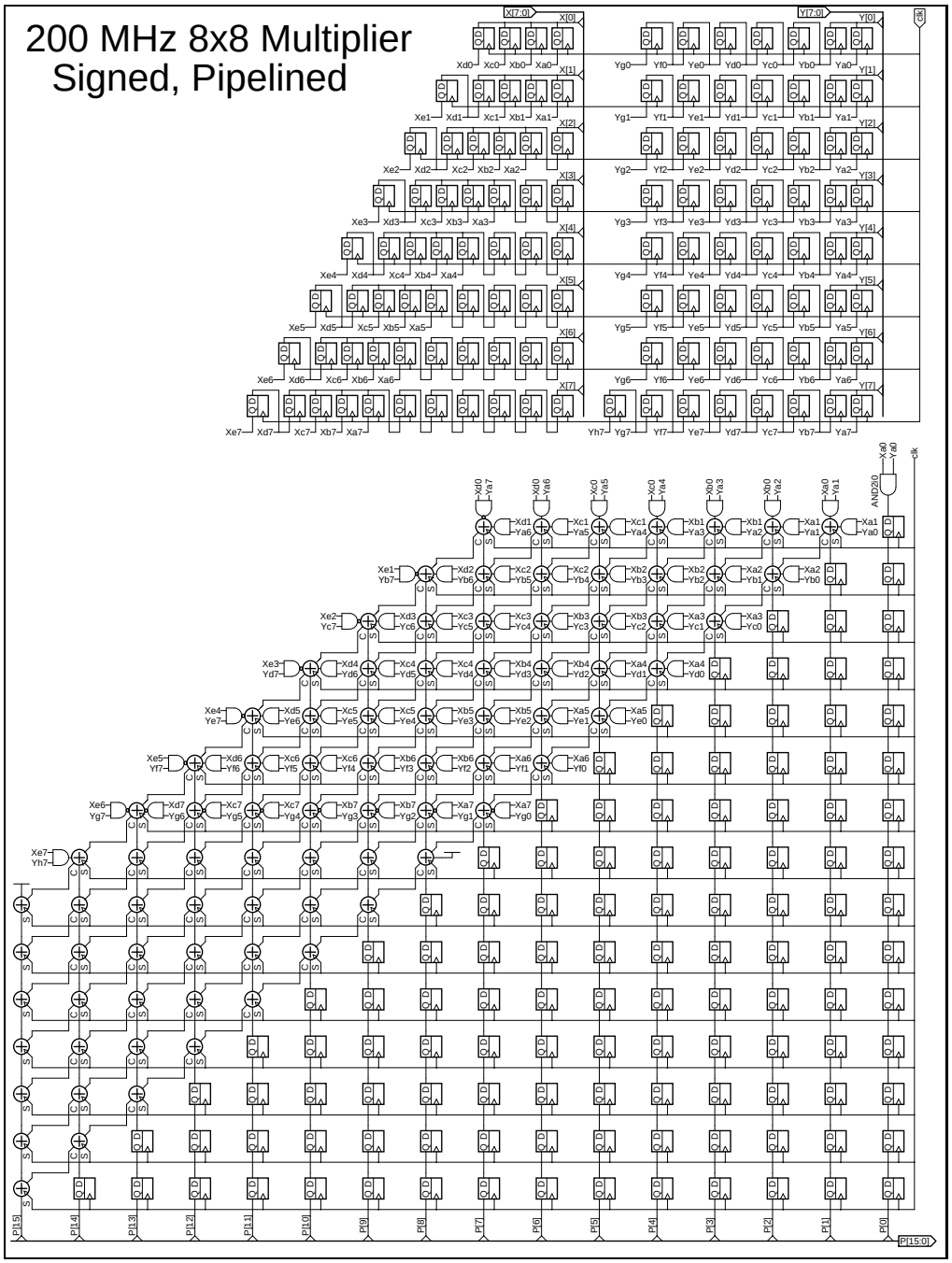

Figure 6: 200MHz Fully Pipelined 8-bit Signed Multiplier

Comparison of Methods Presented

The table below shows the results of worst-case simulations (Vcc = 4.75V, Ta = 70°C) of the QL2009-2PQ208C. All simulations were performed in Quick*Works'* Silos III Verilog simulator with the test fixtures included with the design files. All designs are based on the Baugh-Wooley adder tree algorithm. As with any type of design, the choice of algorithm usually involves a trade-off of some kind: either increased latency vs. reduced throughput or size vs. speed. It is up to the designer to chose the best implementation that meets the design requirement.

Multiplier	Area(Logic Cells)	Speed	Latency
Combinatorial 8-bit Signed (Schematic)	110	25MHz	
Combinatorial 8-bit Signed (HDL)	112	21.7MHz	
1-Stage Pipelined 8-bit Signed(Schematic)	110	35MHz	2 clock cycles
1-Stage Pipelined 8-bit Signed (HDL)	110	34MHz	2 clock cycles
Fully Pipelined 8-bit Signed(Schematic)	390	200MHz	16 clock cycles
Fully Pipelined 8-bit Signed(HDL)	N/A	N/A	

Table 1: Speed and size comparison of the methods presented in this paper

Written By:
John Birkner - V.P. CAE, QuickLogic
Jim Jian - Customer Engineering, QuickLogic
Kevin Smith - Field Applications Engineer, QuickLogic

Further recommended reading

1. J. McCanney and J. McWhirter, "Completely Iterative, Pipelined Multiple Array Suitable for VLSI", IRE Proceedings, Vol. 129. No. 2, April, 1982

2. R. Lyon, "2's Complement Pipelined Multipliers", IEEE Transactions on Communications, April, 1976.

3. Paul J. Song and Giovanni De Micheli "Circuit and Architecture Trade-offs for High-Speed Multiplication", IEEE Journal of Solid-State Circuits, Vol. 26, No. 9, September 1991.

Appendix A: Deriving the Algorithm

For two's compliment numbers A and B defined as:

```
  A = SUM(a[i]2**i for i=0 to m-2) -a[m-1]2**(m-1)
  B = SUM(b[j]2**j for j=0 to n-2) -b[n-1]2**(n-1)
```

the product, P, of A and B is derived as follows:

```
P = AB
  = a[m-1]b[n-1]2**(m+n-2)
  - SUM(a[i]2**i for i=0 to m-2) b[n-1]2**(n-1)
  - SUM(b[j]2**j for j=0 to n-2) a[m-1]2**(m-1)
  + SUM(a[i]b[j]2**(i+j) for i=0 to m-2,for j=0 to n-2)
```

Recognizing that addition is more convenient than subtraction,
and utilizing the equality:

```
  SUM(c[k]2**k for k=0 to n-1)
   = (2**n)-1-SUM(not(c[k]2**k) for k=0 to n-1)
```

then the equation for P can be converted to the following form:

```
P = a[m-1]b[n-1]2**(m+n-2)
  - 2**(n-1)(2**(m-1)-1-SUM(not(a[i]b[n-1])2**i for i=0 to m-2))
  - 2**(m-1)(2**(n-1)-1-SUM(not(b[j]a[m-1])2**j for j=0 to n-2))
  + SUM(a[i]b[j]2**(i+j) for i=0 to m-2,for j=0 to n-2)
```

Simplifying this equation yields:

```
P = a[m-1]b[n-1]2**(m+n-2)
  + SUM(not(a[i]b[n-1])2**(i+n-1) for i=0 to m-2)
  + SUM(not(b[j]a[m-1])2**(j+m-1) for j=0 to n-2)
  + SUM(a[i]b[j]2**(i+j) for i=0 to m-2,for j=0 to n-2)
  + 2**(n-1)
  + 2**(m-1)
  - 2**(m+n-1)
```