Implementing Multipliers with Actel FPGAs

Introduction

Hardware multiplication is a function often required for system applications such as graphics, DSP, and process control. The Actel architecture, which is multiplexer based, allows efficient implementation of multipliers with high performance. Furthermore, the Actel development tools allow the user quickly to create multipliers by using the appropriate algorithm and bit width needed for a specific application. The 1200XL family is the focus of the implementation of this application note, although other Actel families could also be used.

Multiplier Theory

The function of a binary unsigned multiplier, like its decimal counterpart, consists of a multiplicand (X), a multiplier (Y), and a product (P). The result is the product of the multiplier and the multiplicand (P = X * Y). Figure 1 shows the complete multiplication of two four-bit numbers producing an eight-bit product. As in decimal multiplication, the least significant digit of the multiplier combines with each digit of the multiplicand, forming a partial product (Y0X3, Y0X2, Y0X1, Y0X0). Three other partial products are similarly formed. To arrive at the final result, all four of the partial products are added (P7, P6…P0). Note that the most significant bit of the product (P7) is required, due to a possible carry from the other bits.

Conventional Multiplier Algorithm

The conventional approach to implementing a multiplier in digital logic is to AND individual multiplier and multiplicand bits to generate the partial products (PP1, PP2, PP3, PP4). For a four-bit multiplier, this would consist of 16 dual-input AND gates and three adders, as shown in Figure 2. The simplest method to sum the partial products is to have all three adders to be eight bits. Not all of the partial products generate eight bits, so smaller adders could be used. However, tracking which partial sums can be dropped and which need to propagate as carries to the next stage becomes complex and time-consuming, especially with larger bit widths. More important, the conventional multiplier implementation is resource intensive and does not produce optimal performance. Fortunately, another approach is possible.

L-Booth Algorithm Implementation

The L-Booth algorithm employs an alternative technique based on multiplexers, which are an ideal fit for the Actel architecture. For the four-bit implementation, the multiplier's two least significant bits are handled separately from the two most significant bits. Effectively, the multiplexer replaces the first stage of partial sum generation. Figure 3 illustrates the mathematics that explains the L-Booth algorithm. The two least significant multiplier bits (Y1, Y0) are handled separately from the two most significant bits (Y3, Y2). In both cases, the four possible combinations of the multiplier bits are covered with the multiplexer, resulting in the partial products PPA and PPB. Specifically for multiplexer A, the four combinations are zero (the trivial case), X (when Y1=0 and Y0=1), X shifted left or 2X (when Y1=1 and Y0=0), and 3X (when Y1=Y0=1). The multiplexers are eight bits deep to accommodate all eight possible inputs for the adders. To obtain the final product, the two partial sums are added with an eight-bit adder. High-speed adders are used in this implementation since shortest delay from input to output is the primary design constraint. Figure 4 shows the implementation of the L-Booth multiplier. The complete schematic for the fout-bit L-Booth multiplier is shown in Figure 5. Note the use of the five-bit adder to generate the required 3X input for the multiplexers. The name of the schematic is *LBMULT4*, indicating that it uses the L-Booth algorithm.

Multiplicand	>					**X3**	**X2**	**X1**	**X0**
Multiplier	>				x	**Y3**	**Y2**	**Y1**	**Y0**
1st partial product	>					Y0X3	Y0X2	Y0X1	Y0X0
2nd partial product	>				Y1X3	Y1X2	Y1X1	Y1X0	
3rd partial product	>			Y2X3	Y2X2	Y2X1	Y2X0		
4th partial product	>	+	Y3X3	Y3X2	Y3X1	Y3X0			
Final product	>	**P7**	**P6**	**P5**	**P4**	**P3**	**P2**	**P1**	**P0**

Figure 1 • *Four-Bit Binary Multiplication*

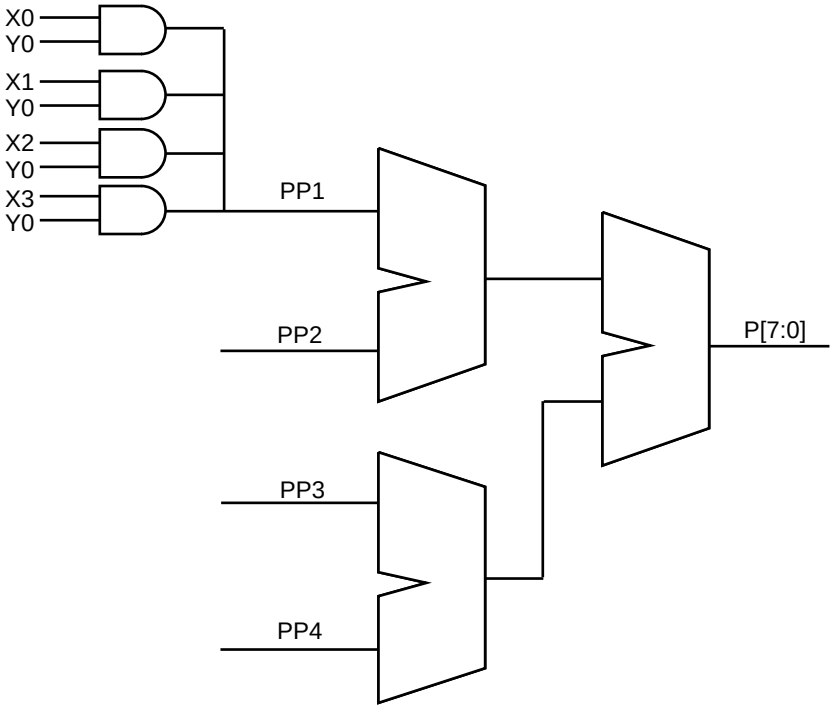

Figure 2 • *Classical Implementation of Four-Bit Multiplier*

						X3	**X2**	**X1**	**X0**
Only 2 LSB used	>				x			**Y1**	**Y0**
if Y1=0,Y0=0		0	0	0	0	0	0	0	0
if Y1=0, Y0=1		0	0	0	0	X3	X2	X1	X0
if Y1=1, Y0=0		0	0	0	X3	X2	X1	X0	0
if Y1=1, Y0=1		0	0	0	X3	X3+X2	X2+X1	X1+X0	0
Multiplexer A result	>	0	0	PPA5	PPA4	PPA3	PPA2	PPA1	PPA0
						X3	**X2**	**X1**	**X0**
Only 2 MSB used	>				x	**Y3**	**Y2**		
if Y3=0,Y2=0		0	0	0	0	0	0	0	0
if Y3=0, Y2=1		0	0	X3	X2	X1	X0	0	0
if Y3=1, Y2=0		0	X3	X2	X1	X0	0	0	0
if Y2=1, Y2=1		0	X3	X3+X2	X2+X1	X1+X0	0	0	0
Multiplexer B result	>	PPB7	PPB6	PPB5	PPB4	PPB3	PPB2	0	0
		0	0	PPA5	PPA4	PPA3	PPA2	PPA1	PPA0
	+	PPB7	PPB6	PPB5	PPB4	PPB3	PPB2	0	0
Final product	>	P7	P6	P5	P4	P3	P2	P1	P0

Figure 3 • *Four-Bit Binary Multiplication Using L-Booth Algorithm*

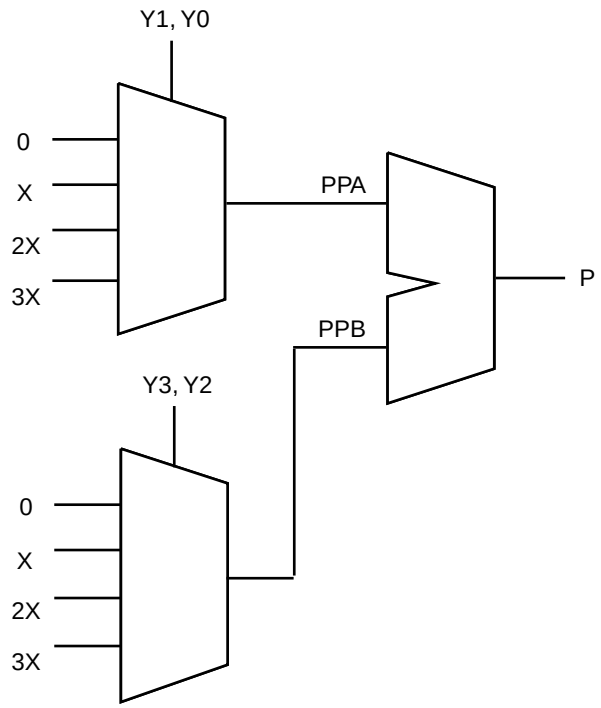

Figure 4 • *L-Booth Multiplier Implementation*

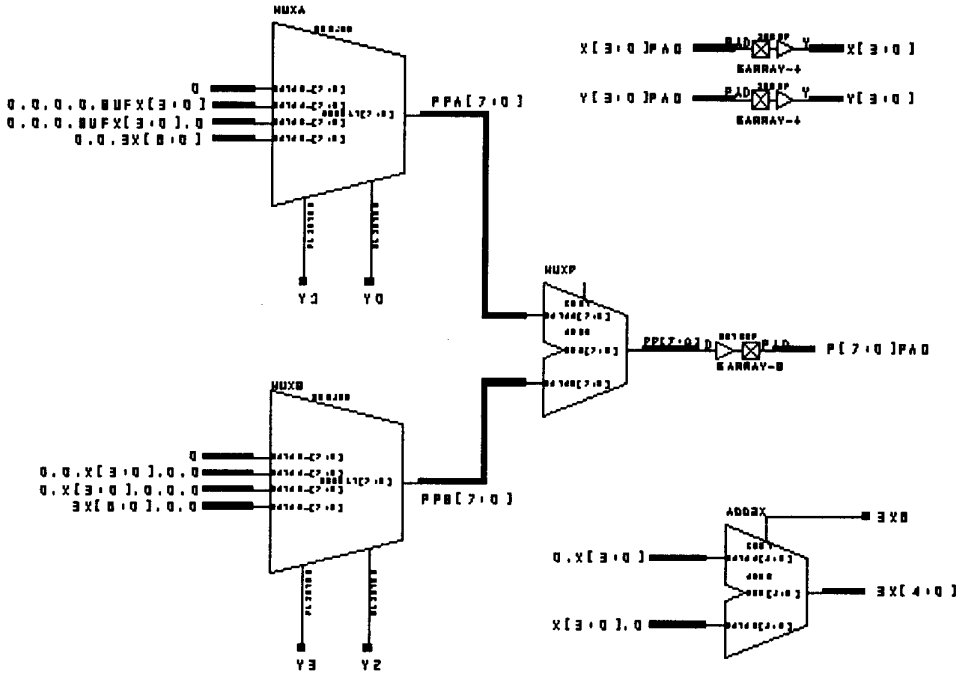

Figure 5 • *Schematic Implementation of Four-Bit L-Booth Multiplier*

Pipelined Multiplier

The previous multiplier implementations were entirely combinatorial. The output product is valid after all input values have propagated through the combinatorial logic. By introducing registers between the levels of logic, the stages of the multiplication can be broken up and synchronized with a clock. By doing so, the effective speed of multiple multiplications is increased, although the result is delayed by the number of register stages that are added. This delay is referred to as the *circuit latency*. Figure 6 shows the pipelined version of the four-bit multiplier, PMULT4.

Two levels of registers are used in the PMULT4 design, resulting in a latency of one cycle (i.e., the result appears one clock cycle later as shown in Figure 7). The distribution of registers is optimal since both stages contain three levels of logic. (The five-bit adder has two levels combined with one for the multiplexer, for a total of three levels. The eight-bit adder has three levels of logic internal.) This means that the multiplication can be done with three levels of combinatorial logic, one register, and data setup. Furthermore, with the 1200XL family, a combinatorial level is absorbed within the sequential module. This means that a four-bit multiplier could be done at a frequency in excess of 60 MHz. (Actual device performance is discussed in detail later in this application note.) The performance can be further increased at the cost of additional registers and circuit latency.

Design Tools

Designing multipliers with the Actel development tools is particularly easy since the basic blocks required (adders, multiplexers, and registers) can be quickly created with the ACTgen Macro Builder. Figure 8 shows the ACTgen main menu with macro category selections. As an example, an eight-bit fast adder with the name of *SAMPLE* will be created with ACTgen. The adder menu in Figure 9 shows the available options: adder variations, bus width, carry in, and carry out. The summary report is shown in Figure 10, and the generated symbol is shown in Figure 11. The multiplexers and registers can be created equally quickly with ACTgen. Using this approach, it is very easy to create multipliers of any bit width by changing the ACTgen parameters. Another modification that can be changed is the type of adder created by ACTgen. By selecting a ripple adder instead of a high-speed one, a more compact multiplier can be created. By making such a change, the four-bit multiplier would require 20 percent fewer modules.

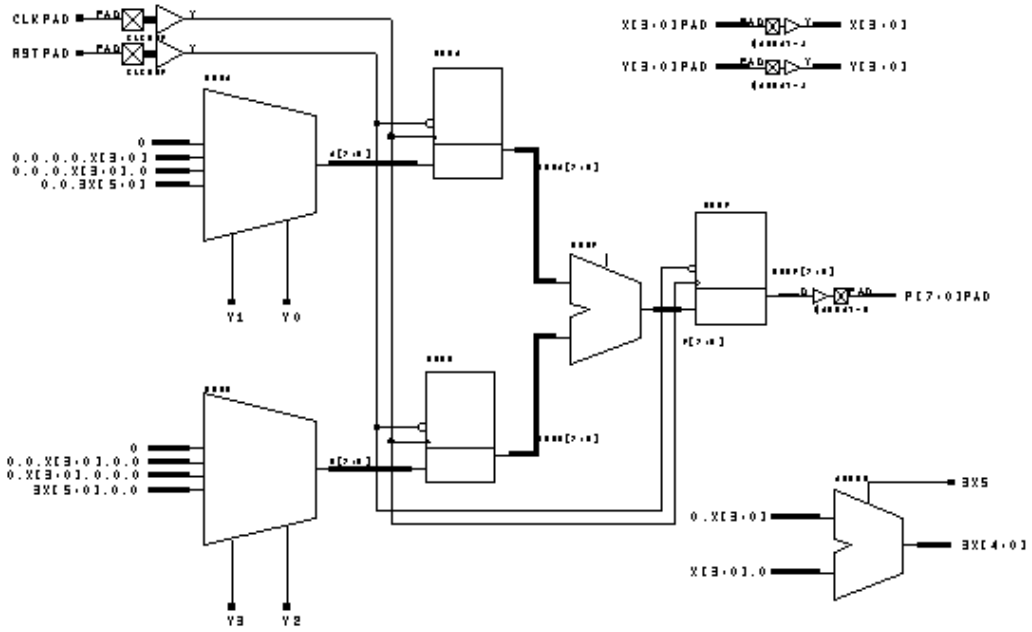

Figure 6 • *Pipelined Four-Bit Multiplier*

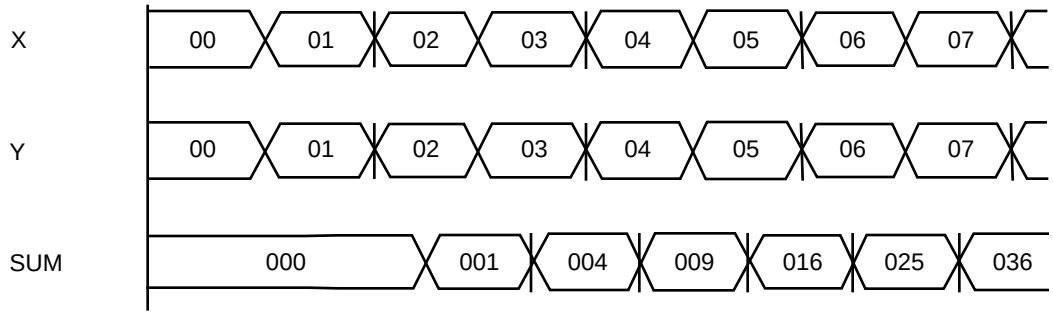

Figure 7 • *Timing Waveform of Pipelined Four-Bit Multiplier with One Cycle Latency*

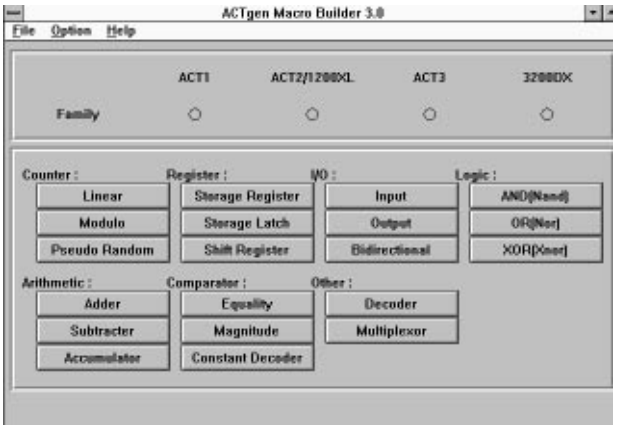

Figure 8 • *ACTgen Macro Builder Main Menu*

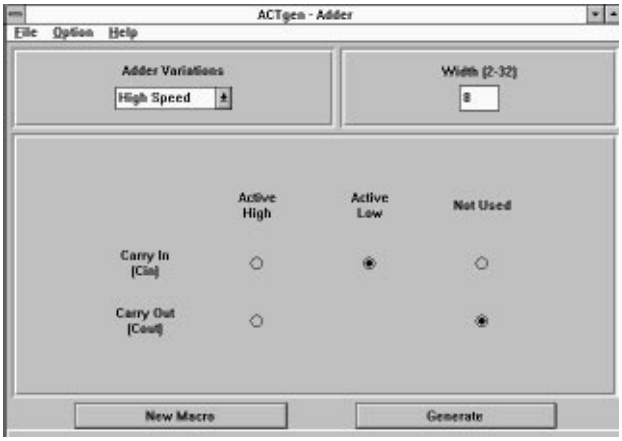

Figure 9 • *ACTgen Macro Builder Adder Menu*

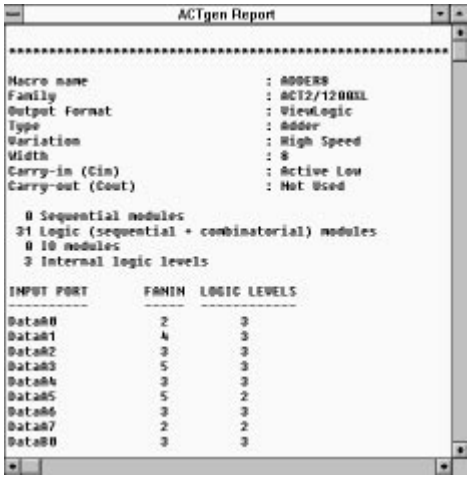

Figure 10 • *ACTgen Macro Builder Summary Report*

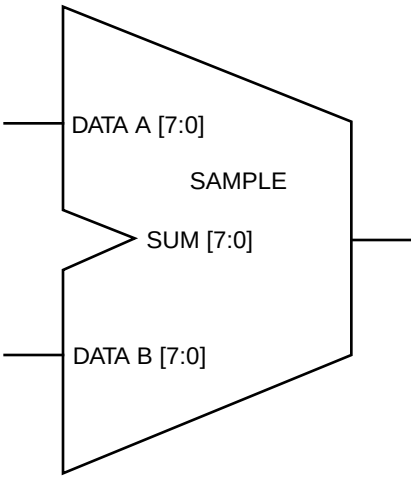

Figure 11 • *ACTgen-Generated Adder Symbol*

Hardware Description Language

Another approach to implementing multipliers is defining the functionality in Hardware Description Language (HDL). Generally, higher-level descriptions are fast to implement but somewhat slower than a manual or macro generator approach. A multiplier is particularly challenging since it is exclusively an arithmetic function, which is typically not well suited to pure synthesis algorithms. However, the Actel ACTmap VHDL Synthesis tool performs very well. The design was optimized for area and required only two modules more than the best manual implementation.

The VHDL source file for a four-bit multiplier is shown in Figure 12. Note how compact the required description is compared with previous schematic implementations. Furthermore, to create an eight-bit, sixteen-bit, or n-bit version would require only changing the BIT_VECTOR definitions.

```
library ASYL;
use ASYL.pkg_arith.all;

entity MULT4 is
        port(X, Y: in BIT_VECTOR(3 downto 0);
                P: out BIT_VECTOR(7 downto 0));
end MULT4;

architecture ARCHI of
        MULT4 is
        begin       P <= X * Y;
end ARCHI;
```

Figure 12 • *VHDL Source Code for Four-Bit Multiplier*

Multiplier Performance

All of the multiplier approaches discussed have been implemented in the Actel 1200XL devices. Table 1 shows all of these including many of the statistics required to make an educated decision on the best approach, depending on design needs. As with most designs, there is almost always a trade-off of system performance and design resources. In the case of the multiplier implementations, this is also true. In fact, there is a monotonic relationship of increasing performance with increasing module count for every multiplier implementation. The speed is obtained by implementing the design for the 1225XL-1 device and obtaining static timer worst-case commercial conditions after place and route. The speed refers to the worst-case path from input pad to output pad for the combinatorial multipliers and the longest internal-clock-to-data path (including data setup time). The "Util" column indicates what percentage of the 1225XL is being used by the multiplier.

Tables 2 and 3 show the statistics for eight-bit and sixteen-bit multipliers, respectively. As before, performance is based on actual placed and routed designs using the Actel static timing analyzer.

Conclusion

Multipliers can be quickly and easily implemented in Actel FPGAs, providing both efficient usage and high performance. The 1200XL family is particularly well suited for the applications. There is a range of options available to the user when designing multipliers: speed/area trade-offs, latency, and design method. Armed with this application information, the Actel development tools, and the 1200XL devices, designers can effectively create multipliers to meet their individual requirements.

Table 1 • *Statistics of Four-Bit Multipliers*

Design	Description	Device	Speed	# Modules	# Levels	Latency	Util
PMULT4	4 by 4 pipelined	1225XL-1	57 MHz	67	3	1 cycle	15%
LBMULT4	4 by 4 comb	1225XL-1	24 MHz	60	6	n/a	13%
PRMULT4	4 by 4 pipelined, ripple adder	1225XL-1	21 MHz	48	10	1 cycle	11%
VHDMULT4	4 by 4 comb, VHDL source	1225XL-1	19 MHz	50	8	n/a	12%
RBMULT4	4 by 4 comb, ripple adder	1225XL-1	14 MHz	48	12	n/a	11%

Table 2 • *Statistics of Eight-Bit Multipliers*

Design	Description	Device	Speed	# Modules	# Levels	Latency	Util
PMULT8	8 by 8 pipelined	1225XL-1	44 MHz	276	3	3 cycles	62%
LBMULT8	8 by 8 comb	1225XL-1	14 MHz	232	10	n/a	52%
PRMULT8	8 by 8 pipelined	1225XL-1	10 MHz	188	17	3 cycles	42%
RBMULT8	8 by 8 comb	1225XL-1	8 MHz	164	22	n/a	37%
VHDMULT8	8 by 8 comb, VHDL source	1225XL-1	8 MHz	325	22	n/a	73%

Table 3 • *Statistics of Sixteen-Bit Multipliers*

Design	Description	Device	Speed	# Modules	# Levels	Latency	Util
PMULT16	16 by 16 pipelined	1280XL-1	28 MHz	1011	4	3 cycles	83%
LDMULT16	16 by 16 comb	1280XL-1	8 MHz	844	16	n/a	69%
PRMULT16	16 by 16 pipelined	1280XL-1	5 MHz	786	33	3 cycles	64%
RBMULT16	16 by 16 comb	1240XL-1	4 MHz	656	40	n/a	96%