Highly Optimized 2-D Convolvers in FLEX Devices

February 1997, ver. 1 | Application Note 82

Introduction

The FLEX® 10K and FLEX 8000 device families can efficiently implement two-dimensional convolvers, which are useful for many image processing applications such as filtering, sharpening, edge detection, and interpolation. A convolver processes information in small pieces, such as a 3×3 convolution window. Each element in the convolution window is multiplied by a constant coefficient, and then the products are summed to obtain the desired result.

This application note describes how to create more efficient convolvers in FLEX 10K and FLEX 8000 designs.

For more information on two-dimensional convolvers, see the following documents:

- *Application Note 73 (Implementing FIR Filters in FLEX Devices)*
- *Product Information Bulletin 21 (Implementing Logic with the Embedded Array in FLEX 10K Devices)*

Conventions

For this application note, the 9 locations in a 3×3 convolution window are numbered for reference, as shown in Figure 1. The subscript indicates which tap or coefficient is being referenced. The variable T_n refers to the tap data value at location n (e.g., T_2 refers to the tap value at location 2).

Figure 1. Tap & Coefficient Reference Locations

1	2	3
4	5	6
7	8	9

A-AN-082-01

Conventional Convolver

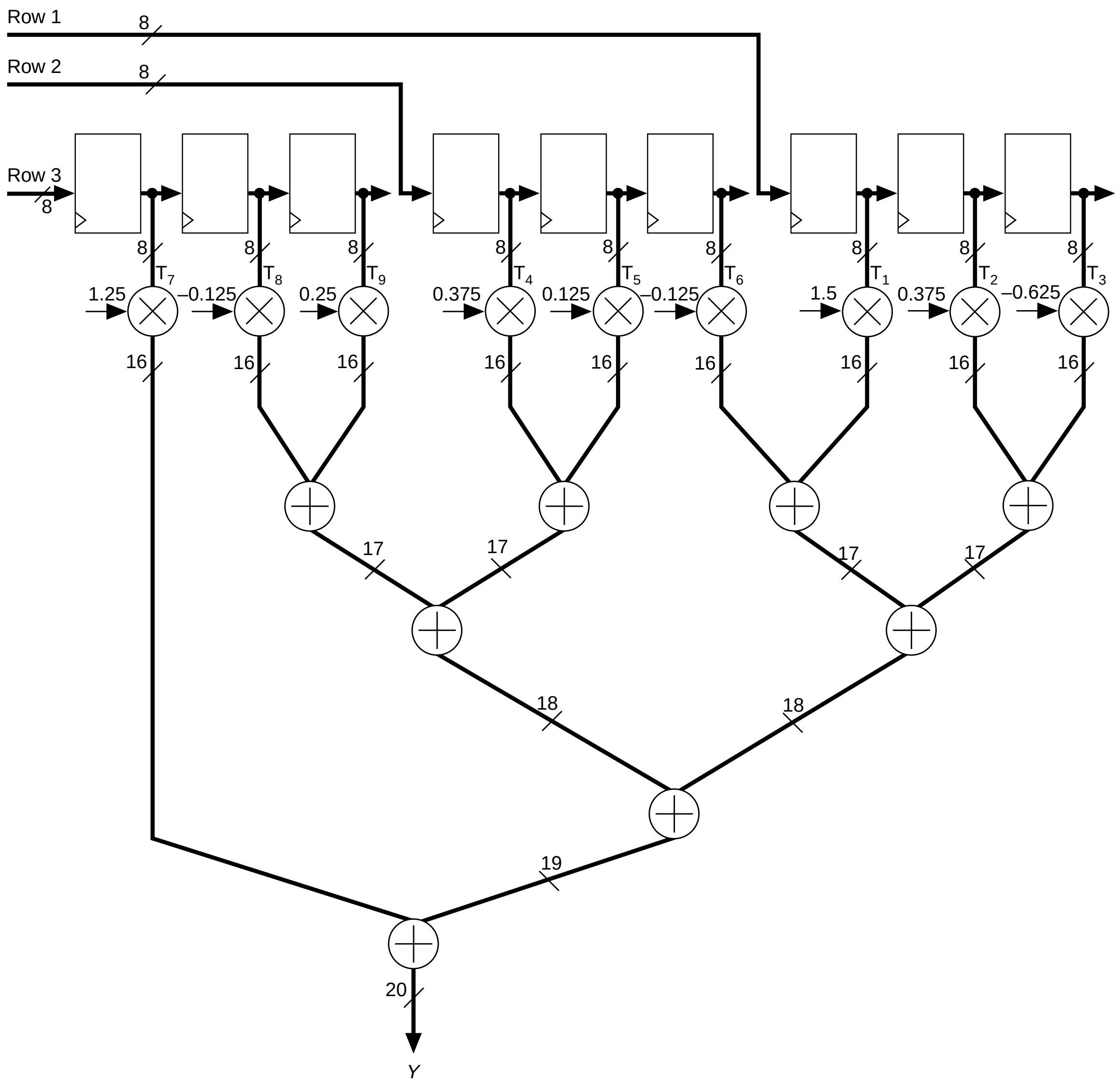
Figure 2 shows the coefficient window for a sample convolver design.

Figure 2. Coefficient Window

Row 1	1.5	0.375	–0.625
Row 2	0.375	0.125	–0.125
Row 3	1.25	–0.125	0.250

Figure 3 shows a block diagram of a conventional convolver using the coefficients shown in Figure 2. In this convolver, the coefficients remain constant and the tap values change every clock cycle as new data is shifted into the convolver.

Figure 3. Conventional Convolver Block Diagram



The output of each register is called a tap (T_n). Each tap is multiplied by the corresponding coefficient (C_n) from Figure 2, and then the products are summed to produce the final result (Y). The equation for the convolver in Figure 3 is shown below:

$$Y = \sum_{1}^{9} T_n C_n$$

Building the conventional two-dimensional convolver for maximum speed requires 9 multipliers and an adder tree consisting of 8 separate adders. Assuming the data and coefficients are 8 bits wide and the coefficients are constant, each multiplier would require 36 logic elements (LEs) and each adder would require one LE per bit, for a total of 539 LEs for the entire design.

Highly Optimized Convolver

You can improve the speed and logic efficiency of the conventional convolver by taking advantage of the numeric relationships between the coefficients. Although the convolution window in Figure 2 is asymmetric, all of the coefficients are integer multiples of 0.125. By rewriting the coefficient window and substituting K = 0.125, you obtain the window shown in Figure 4.

Figure 4. Modified Coefficient Window

12	3	–5
3	1	–1
10	–1	2

× K

In this example, you can take advantage of the fact that all the factored coefficients are integers. Multiplying the taps by the coefficients 12, 3, 5, 1, and 10 is simple because all the integers are the sum of powers of 2. For example, multiplying tap T_1 by 12 is the same as adding $8T_1$ and $4T_1$ (i.e., $12T_1 = 8T_1 + 4T_1$). Because 8 and 4 are both powers of 2, multiplication can be avoided. Instead, T_1 can be shifted left by 3 bits and 2 bits, respectively. Therefore, $12T_1$ can be computed with a single adder that is only 10 bits wide, assuming T_1 is an 8-bit number. For example, if T_1 = 11111111_B, you can calculate $12T_1$ as shown in Figure 5.

Figure 5. Multiplying 12 × T_1

$$\begin{array}{r} 11111111000 \\ +\ 1111111100 \\ \hline 101111110100 \end{array} \quad = \quad \begin{array}{r} 8 \times T_1 \\ +\ 4 \times T_1 \\ \hline 12 \times T_1 \end{array}$$

Because the 2 least significant bits (LSBs) of $8T_1$ and $4T_1$ are always 0, the 2 extra bits in the adder are not required. Therefore, this calculation requires only 10 LEs.

Similarly, 3, 5, and 10 can each be computed with a single adder. Multiplying or dividing by any power of 2 is a shift left or right and does not require any LEs. However, LEs are required to add the product into the final result.

To further improve the logic efficiency of this design, you can take advantage of the “sparseness” of the coeffiecients (i.e., the relatively few 1s in the coefficients). The basic method is to add or subtract various multiples of 2 of the taps to obtain the desired multiplication factor—e.g., add together all taps that have a 1 in the most significant bit (MSB). Table 1 lists each tap’s coefficient and its binary representation. Numbers in parentheses are negative.

Table 1. Binary Representation of the 9 Coefficients

Coefficient	Weight	
	Decimal	Binary
C_1	12	1100
C_2	3	0011
C_3	–5	(0101)
C_4	3	0011
C_5	1	0001
C_6	–1	(0001)
C_7	10	1010
C_8	–1	(0001)
C_9	2	0010

To perform these calculations efficiently in parallel, begin by adding taps with a 1 in the MSB and work your way down to the LSB. In this example, the MSB of C_1 and C_7 is 1. Therefore, you should calculate an intermediate result by adding C_1 and C_7, remembering to shift the product left by 3 bits before adding it to the final result.

Similarly, the (MSB – 1) bit of C_1 and C_3 is 1. In this case, compute $(C_1 - C_3)$, remembering to shift the product left by 2 bits before adding it to the final result.

Repeat this procedure for the next 2 bits of the weight, i.e., the (MSB – 2) bit and the LSB. Next, add all of the products to produce the final result. Last, multiply the final result by K. Because 0.125 is a power of 2, no multiplication is necessary—the final result is simply shifted right by 3 bits.

The following steps show mathematically how to use this method to create a highly optimized convolver design, or any other multiply and accumulate (MAC) design. All numbers are in binary.

$$Y = T_1C_1 + T_2C_2 + T_3C_3 + T_4C_4 + T_5C_5 + T_6C_6 + T_7C_7 + T_8C_8 + T_9C_9$$

1. Substitute the binary values of the coefficients:

$$Y = (T_1 \times 1100) + (T_2 \times 0011) - (T_3 \times 0101) + (T_4 \times 0011) + \\ (T_5 \times 0001) - (T_6 \times 0001) + (T_7 \times 1010) - (T_8 \times 0001) + (T_9 \times 0010)$$

2. Factor out 8 (1000_B):

$$Y = [(T_1 + T_7) \times 1000] + (T_1 \times 100) + (T_2 \times 011) - (T_3 \times 101) + \\ (T_4 \times 011) + (T_5 \times 001) - (T_6 \times 001) + (T_7 \times 010) - (T_8 \times 001) + (T_9 \times 010)$$

3. Factor out 4 (100_B):

$$Y = [(T_1 + T_7) \times 1000] + [(T_1 - T_3) \times 100] + (T_1 \times 00) + (T_2 \times 11) - \\ (T_3 \times 01) + (T_4 \times 11) + (T_5 \times 01) - (T_6 \times 01) + (T_7 \times 10) - (T_8 \times 01) + \\ (T_9 \times 10)$$

4. Factor out 2 (10_B):

$$Y = [(T_1 + T_7) \times 1000] + [(T_1 - T_3) \times 100] + [(T_2 + T_4 + T_7 + T_9) \times 10] + \\ (T_1 \times 0) + (T_2 \times 1) - (T_3 \times 1) + (T_4 \times 1) + (T_5 \times 1) - (T_6 \times 1) + (T_7 \times 0) - \\ (T_8 \times 1) + (T_9 \times 0)$$

5. Add the remaining non-zero terms:

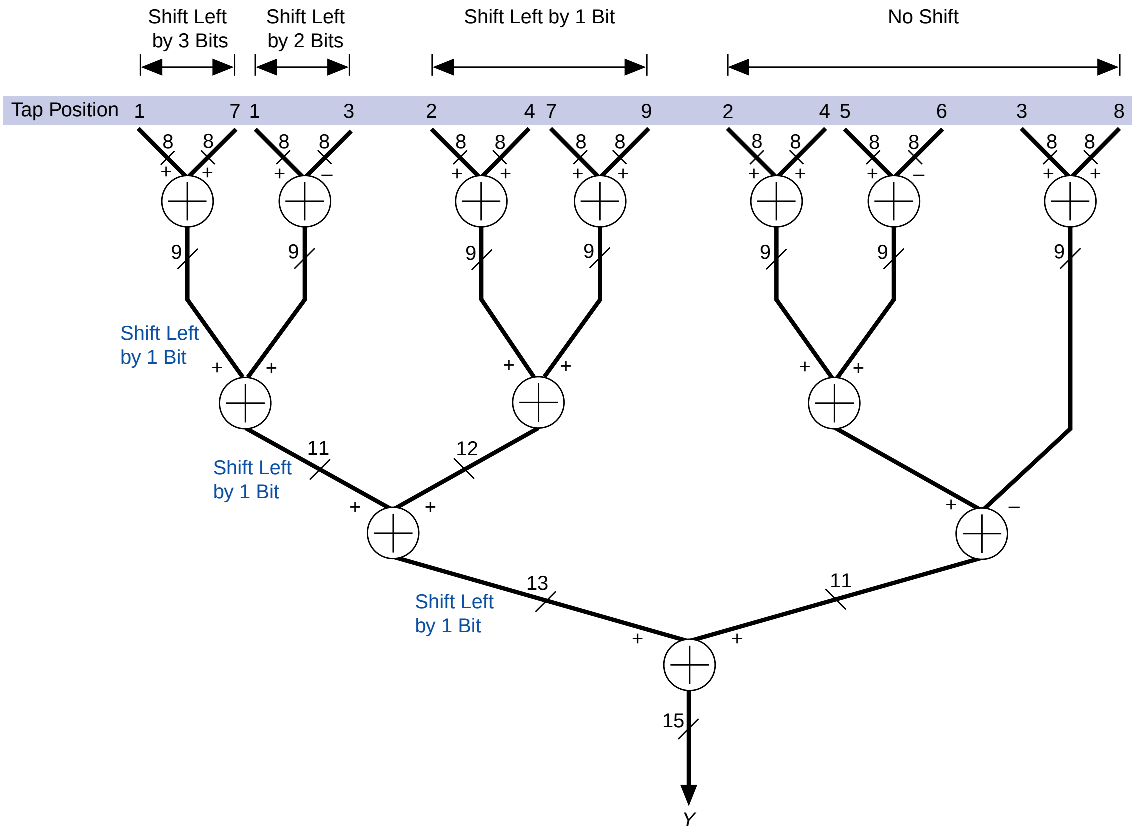
$$Y = [(T_1 + T_7) \times 1000] + [(T_1 - T_3) \times 100] + [(T_2 + T_4 + T_7 + T_9) \times 10] + \\ [(T_2 - T_3 + T_4 + T_5 - T_6 - T_8)] \times 1$$

6. Group the terms by significance and perform more factoring:

$$Y = [(T_1 + T_7) \times 10] + [(T_1 - T_3) \times 1] \times 10 + [(T_2 + T_4 + T_7 + T_9) \times 1 \times 10] + \\ [(T_2 - T_3 + T_4 + T_5 - T_6 - T_8) \times 1]$$

Figure 6 shows a block diagram of the multiply and accumulate section of the optimized convolver.

Figure 6. Highly Optimized Convolver Block Diagram



The highly optimized fixed-coefficient convolver uses 215 LEs, as opposed to the 529 LEs required by the conventional convolver shown in Figure 3—a savings of 60%.

The highly optimized design, written in the Altera Hardware Description Language (AHDL™), is shown in Figure 7. This pipelined design will run at over 90 MHz in a FLEX 8000A device with a -2 speed grade, a speed that is the equivalent of 810 MIPS for only 214 LEs.

☞ This design—named **fstconv**—can be downloaded from the Altera world-wide web site at **http://www.altera.com**.

Figure 7. fstconv.tdf File (Part 1 of 2)

```
SUBDESIGN fstconv
(
     in[3..1][8..1]           : INPUT;
     out[15..1]               : OUTPUT;
     clk                      : INPUT;
)
VARIABLE
     row1[3..1][8..1]         : DFF; -- Row 1 of shift register
     row2[3..1][8..1]         : DFF; -- Row 2 of shift register
     row3[3..1][8..1]         : DFF; -- Row 3 of shift register

     inter_sh3[9..1]          : DFF; -- Intermediate shift by 3 node
     inter_sh2[9..1]          : DFF; -- Intermediate shift by 2 node
     inter_sh1_t[2..1][9..1]  : DFF; -- Shift by 1 node
     inter_sh1[10..1]         : DFF; -- More shift by 1 node
     inter_sh0_t1[3..1][9..1]: DFF; -- Shift by 0
     inter_sh0_t2a[10..1]     : DFF;
     inter_sh0_t2b[9..1]      : DFF;
     inter_sh0[11..1]         : DFF;
     inter_3_2[11..1]         : DFF; -- Shift by 3 + shift by 2
     inter_3_2_1[13..1]       : DFF; -- inter_3_2 + shift by 1
     inter_3_2_1_0[15..1]     : DFF; -- inter_3_2_1 + shift by 0
BEGIN

-- Connect the clock to all the internal registers
     (inter_sh3[].clk, inter_sh2[].clk, inter_sh1_t[][].clk,
     inter_sh1[].clk, inter_sh0_t1[][].clk,
     inter_sh0_t2a[].clk, inter_sh0_t2b[].clk,
     inter_sh0[11..1].clk, inter_3_2[].clk, inter_3_2_1[].clk
     inter_3_2_1_0[].clk) = clk;

-- Connect the shift register:
-- Data will shift from left to right, which will be 1 to 3
     row1[1][] = in[1][];
     row2[1][] = in[2][];
     row3[1][] = in[3][];
     row1[3..2][] = row1[2..1][];
     row2[3..2][] = row2[2..1][];
     row3[3..2][] = row3[2..1][];
     row1[][].clk = clk;
     row2[][].clk = clk;
     row3[][].clk = clk;
```

Figure 7. fstconv.tdf (Part 2 of 2)

```
-- The shift by 3 data
     inter_sh3[]      = (row1[1][8], row1[1][]) + (row3[1][8], row3[1][]);

-- The shift by 2 data
     inter_sh2[]      = (row1[1][8], row1[1][]) - (row1[3][8], row1[3][]);

-- The shift by 1 data
     inter_sh1_t[1][] = (row1[2][8], row1[2][]) +
                        (row2[1][8], row2[1][]); -- 2 + 4
     inter_sh1_t[2][] = (row3[1][8], row3[1][]) +
                        (row3[3][8], row3[3][]); -- 7 + 9
     inter_sh1[]      = (inter_sh1_t[1][9], inter_sh1_t[1][]) +
                        (inter_sh1_t[2][9], inter_sh1_t[2][]);

-- The shift by zero data
     inter_sh0_t1[1][]= (row1[2][8], row1[2][]) +
                        (row2[1][8], row2[1][]);
     inter_sh0_t1[2][]= (row2[2][8], row2[2][]) -
                        (row2[3][8], row2[3][]);
     inter_sh0_t1[3][]= (row1[3][8], row1[3][]) +
                        (row3[2][8], row3[2][]);
     inter_sh0_t2a[]  = (inter_sh0_t1[1][9], inter_sh0_t1[1][]) +
                        (inter_sh0_t1[2][9], inter_sh0_t1[2][]);
     inter_sh0_t2b[]  = inter_sh0_t1[3][];
     inter_sh0[]      = (inter_sh0_t2a[10], inter_sh0_t2a[]) -
                        (inter_sh0_t2b[9], inter_sh0_t2b[9],
                        inter_sh0_t2b[]);

-- Shift by 3 + shift by 2
     inter_3_2[]      = (inter_sh3[9], inter_sh3[], gnd) +
                        (inter_sh2[9], inter_sh2[9], inter_sh2[]);

-- Shift by 3 + shift by 2 + shift by 1
     inter_3_2_1[]    = (inter_3_2[11], inter_3_2[], gnd) +
                        (inter_sh1[10], inter_sh1[10], inter_sh1[10],
                        inter_sh1[]);

-- Shift by 3 + 2 + 1 + 0
     inter_3_2_1_0[]  = (inter_3_2_1[13], inter_3_2_1[], gnd) +
                        (inter_sh0[11], inter_sh0[11],
                        inter_sh0[11], inter_sh0[11], inter_sh0[]);

-- Output
     out[] =            inter_3_2_1_0[];
END;
```

Conclusion

This application note shows how to improve the logic efficiency of a convolver design when the coefficients can be factored and are sparse. To reduce the amount of logic required for your convolver design, use the following tips:

- If possible, factor out a common coefficient, so that a single final multiplication can be performed at the end (if the factor is a multiple of 2, multiplication is not required).
- If possible, set all the coefficients to a power of 2 or to the sum of a few powers of 2. Logic is not required to multiply a tap by the coefficients 1, 2, 4, and 8—the taps are just shifted left by 0, 1, 2, or 3 bits, respectively. Likewise, the numbers 3, 5, 6, 9, 10, and 12 can be calculated with a single adder. The other numbers below 16 will require more logic, but will still be more efficient than using the conventional method.

Notes:

2610 Orchard Parkway
San Jose, CA 95134-2020
(408) 894-7000
http://www.altera.com
Applications Hotline:
(800) 800-EPLD
Customer Marketing:
(408) 894-7104
Literature Services:
(888) 3-ALTERA
lit_req@altera.com

I.S. EN ISO 9001