Research Article

THE ROLE OF DISTRIBUTED ARITHMETIC IN FPGA BASED SIGNAL PROCESSING

D. U. Shah, C. H. Vithlani

Address for Correspondence

1 Asstt Prof., Department of Electronics & Comm. Engineering, R.K.CET, Rajkot, Gujarat, India.
2 Professor, Department of Electronics & Comm. Engineering, GEC, Rajkot, Gujarat, India

ABSTRACT

Distributed Arithmetic (DA) plays a key role in embedding DSP functions in the Xilinx 4000 family of FPGA devices. In this paper the DA algorithm is derived and examples are offered that illustrate its effectiveness in producing gate-efficient designs.

DISTRIBUTED ARITHMETIC REVISITED:

Distributed Arithmetic, along with Modulo Arithmetic, are computation algorithms that perform multiplication with look-up table based schemes. Indeed, DA specifically targets the sum of products (sometimes referred to as the vector dot product) computation that covers many of the important DSP filtering and frequency transforming functions. Ironically, many DSP designers have never heard of this algorithm. Inspired by the potential of the Xilinx FPGA look-up table architecture, the DA algorithm was resurrected and shown to produce very efficient filter designs. The derivation of the DA algorithm is extremely simple but its applications are extremely broad. The mathematics includes a mix of Boolean and ordinary algebra and requires no prior preparation, even for the logic designer.

DISTRIBUTED ARITHMETIC AT A GLANCE:

The arithmetic sum of products that defines the response of linear, time-invariant networks can be expressed as:

\[ y(n) = \sum_{k=1}^{K} A_k x_k(n) \]

where:

- \( y(n) \) is the response of network at time \( n \).
- \( x_k(n) \) is the \( k \)th input variable at time \( n \).
- \( A_k \) is the weighting factor of the \( k \)th input variable that is constant for all \( n \), and so it remains time-invariant.

In filtering applications the constants, \( A_k \), are the filter coefficients and the variables, \( x_k \), are the prior samples of a single data source (for example, an analog to digital converter). In frequency transforming whether the discrete Fourier or the fast Fourier transform - the constants are the sine/cosine basis functions and the variables are a block of samples from a single data source. Examples of multiple data sources may be found in image processing.

The multiply-intensive nature of eq. 1 can be appreciated by observing that a single output response requires the accumulation of \( K \) product terms. In DA the task of summing product terms is replaced by table look-up procedures that are easily implemented in the Xilinx configurable logic block (CLB) look-up table architecture. We start by defining the number format of the variable to be 2’s complement, fractional - a standard practice for fixed-point microprocessors in order to bound number growth under multiplication. The constant factors, \( A_k \), need not be so restricted, nor are they required to match the data word length, as is the case for the microprocessor. The constants may have a mixed integer and fractional format; they need not be defined at this time. The variable, \( x_k \), may be written in the fractional format as shown in eq. 2

\[ x_k = x_{k0} 2^b + \sum_{b=1}^{b-1} x_{kb} 2^b \]

where \( x_{kb} \) is a binary variable and can assume only values of 0 and 1. A sign bit of value -1 is indicated by \( x_{k0} \). Note that the time index, \( n \), has been dropped since it is not needed to continue the derivation. The final result is obtained by first substituting eq.2 into eq.1

\[ y = \sum_{i=1}^{E} \sum_{k=1}^{K} A_k \left( x_{k0} 2^b + \sum_{b=1}^{b-1} x_{kb} 2^b \right) = \sum_{b=1}^{b=1} x_{k0} A_k - \sum_{b=1}^{b-1} x_{kb} 2^b A_k \]

and then explicitly expressing all the product terms under the summation symbols:

\[ y = \sum_{i=1}^{E} \sum_{k=1}^{K} A_k \left( x_{k0} 2^b + \sum_{b=1}^{b-1} x_{kb} 2^b \right) = \sum_{b=1}^{b=1} x_{k0} A_k - \sum_{b=1}^{b-1} x_{kb} 2^b A_k \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]
Each term within the brackets denotes a binary AND operation involving a bit of the input variable and all the bits of the constant. The plus signs denote arithmetic sum operations. The exponential factors denote the scaled contributions of the bracketed pairs to the total sum. We can now construct a look-up table that can be addressed by the same scaled bit of all the input variables and can access the sum of the terms within each pair of brackets. Such a table is shown in fig. 1 and will henceforth be referred to as a Distributed Arithmetic look-up table or DALUT. The same DALUT can be time-shared in a serially organized computation or can be replicated B times for a parallel computation scheme, as described later.

**DALUT Addressing**

<table>
<thead>
<tr>
<th>$x_{0b}$</th>
<th>$A_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{1b}$</td>
<td>$A_1$</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$x_{kb}$</td>
<td>$A_k$</td>
</tr>
</tbody>
</table>

$A_k$ is included in the sum when $2^{k-1}$

![DALUT Addressing Diagram]

**DALUT Content**

<table>
<thead>
<tr>
<th>Address</th>
<th>$A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>$A_0$</td>
</tr>
<tr>
<td>2</td>
<td>$A_1$</td>
</tr>
<tr>
<td>...</td>
<td>$A_2$</td>
</tr>
<tr>
<td>5</td>
<td>$A_4 + A_1$</td>
</tr>
<tr>
<td>6</td>
<td>$A_6 + A_2$</td>
</tr>
<tr>
<td>7</td>
<td>$A_7 + A_1 + A_2$</td>
</tr>
<tr>
<td>8</td>
<td>$A_8$</td>
</tr>
</tbody>
</table>

If $x_{0b}$ is least significant address bit

$A_k$ may be bipolar

For $B = 16$, eq. 4 becomes:

$$y = \sum_{k} \left[ \sum_{i=0}^{B-1} A_k \right]$$

The decomposition of Eq. 5 into an array of two input adders is given below:

$$y = -\left[ \sum_{i=0}^{14} \right] + \left[ \sum_{i=15}^{25} \right]$$

The arithmetic operations have now been reduced to addition, subtraction, and binary scaling. With scaling by negative powers of 2, the actual implementation entails the shifting of binary coded data words toward the least significant bit and the use of sign extension bits to maintain the sign at its normal bit position. The hardware implementation of a binary full adder (as is done in the CLBs) entails two operands, the addend and the augend to produce sum and carry output bits. The multiple bit-parallel additions of the DALUT outputs expressed in eq. 4 can only be performed with a single parallel adder if this adder is time-shared. Alternatively, if simultaneous addition of all DALUT outputs is required, an array of parallel adders is required. These opposite goals represent the classic speed-cost tradeoff.

**THE SPEED TRADEOFF:**

Any new device that can be software configured to perform DSP functions must contend with the well entrenched standard DSP chips, i.e. the programmable fixed point microprocessors that feature concurrently operating hardware multipliers and address generators, and on-chip memories. The first challenge is speed. If the FPGA doesn’t offer higher speed why bother. For a single filter channel the bother is worth it - particularly as the filter order increases. And the FPGA advantage grows for multiple filter channels.

Alas, a simple advantage may not be persuasive in all cases - an overwhelming speed advantage may be needed for FPGA acceptance. It is possible to reach 50 megasamples/sec data sample rates but at a high cost in gate resources. The first two examples will show the end points of the serial/parallel tradeoff continuum.

**THE ULTIMATE IN SPEED:**

Conceivably, with a fully parallel design the sample speed could match the system clock rate. This is the case where all the add operations of the bracketed values (the DALUT outputs) of eq. 4 are performed in parallel. We can gain implementation guidance by rephrasing eq. 4, and to facilitate this process let us abbreviate the contents within each bracket pair by the data bit position. Thus

$$[x_{12} \cdot A_1 + x_{22} \cdot A_2 + \ldots + x_{K2} \cdot A_K]$$

reduces to

$$[\text{sum 2}]$$

and, similarly,

$$[x_{(B-1)2} \cdot A_1 + x_{(B-1)2} \cdot A_2 + \ldots + x_{(B-1)K} \cdot A_K]$$

reduces to

$$[\text{sum(B-1)}]$$

For $B = 16$, eq. 4 becomes:

$$y = -\left[ \sum_{i=0}^{14} \right] + \left[ \sum_{i=15}^{25} \right]$$

The decomposition of Eq. 5 into an array of two input adders is given below:
Equations 5 and 6 are computationally equivalent, but equ. 6 can be mapped in a straight-forward way into a binary tree-like array of summing nodes with scaling effected by signal routing as shown in fig. 2. Each of the 15 nodes represents a parallel adder, and while the computation may can yield responses that include both the double precision (B+C bits) of the implicit multiplication and the attendant processing gain, these adders can be truncated to produce single precision (B bits) responses.

![Figure 2: Example of Fully Parallel DA Model (K=16, B=16)](image)

All B bits of all K data sources must be present to address the B DALUTs. A BxK array of flip-flops is required. Each of the B identical DALUTs contains 2K words with C bits per word where C is the “cumulative” coefficient accuracy. The data flow from the flip-flop array can be all combinatorial; the critical delay path for B=16 is not inordinately long - signal routing through 5 CLB stages and a carry chain embracing 2C adder stages. A system clock in the 10 MHz range may work. Certainly with internode pipelining a system clock of 50 MHz appears feasible. The latency would, in many cases be acceptable; however, it would be problematic in feedback networks (e.g., IIR filters).

THE ULTIMATE IN GATE EFFICIENCY:
The ultimate in gate efficiency would be a single DALUT, a single parallel adder, and, of course, fewer flip-flops for the input data source. Again with our B=16 example, a rephrasing of eq.4 yields the desired result:

\[
y = -[\text{sum0}]+[\text{sum1}]z^2 + ([\text{sum2}]-[\text{sum3}])z^4
\]

\[
+ ([\text{sum4}]+[\text{sum5}])z^6 + ([\text{sum6}]-[\text{sum7}])z^8 + \ldots
\]

\[
+ ([\text{sum8}]-[\text{sum9}]z^14 + [\text{sum10}]-[\text{sum11}])z^{16}
\]

\[
+ ([\text{sum12}]-[\text{sum13}])z^{18} + ([\text{sum14}]-[\text{sum15}])z^{20} + \ldots z^{2B}
\]

Starting from the least significant end, i.e. addressing the DALUT with the least significant bit of all K input variables the DALUT contents, [sum15], are stored, scaled by 2-1 and then added to the DALUT contents, [sum14] when the address changes to the next-to-the-least-significant bits. The process repeats until the most significant bit addresses the DALUT, [sum0]. If this is a sign bit a subtraction occurs. Now a vision of the hardware emerges. A serial shift register, B bits long, for each of the K variables addresses the DALUT least significant bit first. At each shift the output is applied to a parallel adder whose output is stored in an accumulator register. The accumulator output - scaled by 2-1 is the second input to the adder. Henceforth, the adder, register and scalar shall be referred to as a scaling accumulator. The functional blocks are shown in fig. 3. All can be readily mapped into the Xilinx 4000 CLBs. There is a performance price to be paid for this gate efficiency - the computation takes at least B clocks.

BETWEEN THE EXTREMES:
While there are a number of speed-gate count tradeoffs that range from one bit per clock (the ultimate in gate efficiency) to B bits per clock (the ultimate in speed) the question of their effectiveness under architectural constraints remains.
Again consider the case of $B = 16$ and rephrasing eq. 5:

$$y = \sum_{i=0}^{15} \{\text{sum}i\} 2^i$$

where $\{\text{sum}i\}$ are the terms within the rectangular brackets. These terms are stored in a common DALUT which can also serve as $\text{sum}0$ and $\text{sum}15$. Note that the computation takes $B/2 + 1$ or 9 clock periods. The functional blocks of the data path are shown in Fig. 4a.

Figure 4a: Two-bit-at-a-time Distributed Arithmetic Data Path ($B=16$, $K=16$)

There is another way of rephrasing or partitioning eq. 5 that maintains the $B$ clock computation time:

$$y = \left(\sum_{i=0}^{15} \{\text{sum}i\} 2^i\right) 2^{16}$$

Here two identical DALUTs, two scaling accumulators, and a post-accumulator adder (Fig. 4b) are required. While the adder in the scaling accumulator may be single precision, the second adder stage may be double precision to meet performance requirements.

Figure 4b: Two-bit-at-a-time Distributed Arithmetic Data Path ($B=16$, $K=16$)
Consider a third rephrasing of eq. 5.

\[
y = \left[ -(\text{sum}0) + (\text{sum}1)2^{-1} \right] 2^{-6} \\
+ \left[ (\text{sum}2) + (\text{sum}3)2^{-1} \right] 2^{-2} \\
+ \left[ (\text{sum}4) + (\text{sum}5)2^{-1} \right] 2^{-4} \\
+ \left[ (\text{sum}6) + (\text{sum}7)2^{-1} \right] 2^{-6} \\
+ \left[ (\text{sum}8) + (\text{sum}9)2^{-1} \right] 2^{-8} \\
+ \left[ (\text{sum}10) + (\text{sum}11)2^{-1} \right] 2^{-10} \\
+ \left[ (\text{sum}12) + (\text{sum}13)2^{-1} \right] 2^{-12} \\
+ \left[ (\text{sum}14) + (\text{sum}15)2^{-1} \right] 2^{-14}
\]

Here the inner brackets denote a DALUT output while the larger, outer brackets denote the scaling of the scaling accumulator. Two parallel odd-even bit data paths are indicated (fig.4c) with two identical DALUTs. The DALUT addressed by the even bits has its output scaled by \(2^{-1}\) and then is applied to the parallel adder. The adder sum is then applied to the scaling accumulator which yields the desired response, \(y(n)\).

The table indicates that for a 16 bit distributed arithmetic process, the single bit-at-a-time computation can accommodate 10 and, possibly, 12 input variables. For the 22K two bit-at-a-time DALUT organization fewer than 6 input variables can be handled by the XC4025 FPGA, while for the 2x2K scheme 9 input variables may be possible. The distributed arithmetic computation can be configured and shaped to overcome many architectural constraints. Remember the FPGA was not designed with signal processing in mind. Yet with a good understanding of the DA computation process, with an appreciation of the constraints inherent in the CLB and the interconnect environment, and with good knowledge of the DSP algorithm of interest, an FPGA-embedded design can offer the system designer a very attractive alternative to more convention solutions.

REFERENCES

1. Mintzer, L. “FIR filters with the Xilinx FPGA ‘FPGA ’92 ACM/SIGDA Workshop on FPGAs pp. 129-134