Introduction to Distributed Arithmetic

K. Sridharan, IIT Madras

Distributed Arithmetic (DA)

- An efficient technique for calculation of **inner product** or **multiply and accumulate** (MAC)
- The MAC operation is common in Digital Signal Processing Algorithms

What is the direct method of implementing inner products or MAC ?

- The direct method involves using dedicated multipliers
- Multipliers are fast but they consume considerable hardware

An Illustration of MAC Operation

- The following expression represents a multiply and accumulate operation

$$y = A_1 \times x_1 + A_2 \times x_2 + \cdots + A_K \times x_K$$

$$i.e. \quad y = \sum_{k=1}^{K} A_k x_k$$

- A numerical example

$$A = [32, 42, 45, 23] \quad x = [42, 20, -22, 67] \quad (K = 4)$$

$$y = 32 \times 42 + 45 \times 20 + 78 \times (-22) + 23 \times 67$$

$$y = 1344 + 900 - 1716 + 1541 = 2069$$

How does distributed arithmetic work?

- Distributed Arithmetic (DA) is a technique that is **bit-serial** in nature. It can therefore appear to be slow
- It turns out that when the number of elements in a vector is nearly the same as the wordsize, DA is quite fast
- DA `replaces’ the explicit multiplications by **ROM look-ups** → an efficient technique to implement on Field Programmable Gate Arrays (FPGAs)

What does DA achieve ?

- In DA, multiplications are reordered and mixed such that the arithmetic becomes “distributed” through the structure rather than being “lumped”
- Area savings from using DA can be up to 80% in DSP hardware designs
- While distributed arithmetic technique itself has been around for more than 30 years (Peled and Liu, 1974), interest in this has been revived by the use of Field Programmable Gate Arrays (FPGAs) for DSP
- When DA is implemented in FPGAs, one can take advantage of memory in FPGAs to implement the MAC operation

The Formulation for DA

- Consider

$$y = \sum_{k=1}^{K} A_k x_k \qquad \ldots(1)$$

  - a. Let x_k be an N-bit scaled two's complement number. In other words,

$$|x_k| < 1$$

$$x_k : \{b_{k0}, b_{k1}, b_{k2}\ldots\ldots, b_{k(N-1)}\}$$

    where b_{k0} is the *sign bit*
  - b. We can express x_k as $x_k = -b_{k0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n} \qquad \ldots(2)$
  - c. Substituting (2) in (1),

$$y = \sum_{k=1}^{K} A_k \left[-b_{k0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n} \right]$$

$$y = -\sum_{k=1}^{K} (b_{k0} \bullet A_k) + \sum_{k=1}^{K} \sum_{n=1}^{N-1} (A_k \bullet b_{kn}) 2^{-n} \qquad \ldots(3)$$

Continuing the DA formulation …

$$y = -\sum_{k=1}^{K}\left(b_{k0} \bullet A_k\right) + \sum_{k=1}^{K} \underbrace{\left[\sum_{n=1}^{N-1}\left(b_{kn} \bullet A_k\right)2^{-n}\right]}_{\text{Expanding this part}} \quad \ldots(3)$$

$$y = -\underbrace{\sum_{k=1}^{K}\left(b_{k0} \bullet A_k\right)}+\underbrace{\sum_{k=1}^{K}\left[\left(A_k \bullet b_{k1}\right)2^{-1} + \left(A_k \bullet b_{k2}\right)2^{-2} + \cdots + \left(A_k \bullet b_{k(N-1)}\right)2^{-(N-1)}\right]}$$

$$\begin{aligned}
y = &-\left[b_{10} \bullet A_1 + b_{20} \bullet A_2 + \cdots + b_{K0} \bullet A_K\right] \\
&+\left[\left(b_{11} \bullet A_1\right)2^{-1} + \left(b_{12} \bullet A_1\right)2^{-2} + \cdots + \left(b_{1(N-1)} \bullet A_1\right)2^{-(N-1)}\right] \\
&+\left[\left(b_{21} \bullet A_2\right)2^{-1} + \left(b_{22} \bullet A_2\right)2^{-2} + \cdots + \left(b_{2(N-1)} \bullet A_2\right)2^{-(N-1)}\right] \\
&\qquad\vdots \\
&+\left[\left(b_{K1} \bullet A_K\right)2^{-1} + \left(b_{K2} \bullet A_K\right)2^{-2} + \cdots + \left(b_{K(N-1)} \bullet A_K\right)2^{-(N-1)}\right]
\end{aligned}$$

Further simplification leads to

$$
\begin{aligned}
y = -\big[b_{10} \bullet A_1 \quad &+ b_{20} \bullet A_2 \quad + \cdots + \quad b_{K0} \bullet A_K\big] \\
+ \big[(b_{11} \bullet A_1)2^{-1} &+ (b_{12} \bullet A_1)2^{-2} + \cdots + (b_{1(N-1)} \bullet A_1)2^{-(N-1)}\big] \\
+ \big[(b_{21} \bullet A_2)2^{-1} &+ (b_{22} \bullet A_2)2^{-2} + \cdots + (b_{2(N-1)} \bullet A_2)2^{-(N-1)}\big] \\
&\vdots \\
+ \big[(b_{K1} \bullet A_K)2^{-1} &+ (b_{K2} \bullet A_K)2^{-2} + \cdots + (b_{K(N-1)} \bullet A_K)2^{-(N-1)}\big]
\end{aligned}
$$

$$
\begin{aligned}
y = -\big[&b_{10} \bullet A_1 + b_{20} \bullet A_2 + \cdots + b_{K0} \bullet A_K\big] \\
&+ \big[(b_{11} \bullet A_1) + (b_{21} \bullet A_2) + \cdots + (b_{K1} \bullet A_K)\big]2^{-1} \\
&+ \big[(b_{12} \bullet A_1) + (b_{22} \bullet A_2) + \cdots + (b_{K2} \bullet A_K)\big]2^{-2} \\
&\vdots \\
&+ \big[(b_{1(N-1)} \bullet A_1) + (b_{2(N-1)} \bullet A_2) + \cdots + (b_{K(N-1)} \bullet A_K)\big]2^{-(N-1)}
\end{aligned}
$$

The rewrite showing interchange of sum ..

$$
\begin{aligned}
y = & -\left[b_{10} \bullet A_1 + b_{20} \bullet A_2 + \cdots + b_{K0} \bullet A_K\right] \\
& + \left[(b_{11} \bullet A_1) + (b_{21} \bullet A_2) + \cdots + (b_{K1} \bullet A_K)\right] 2^{-1} \\
& + \left[(b_{12} \bullet A_1) + (b_{22} \bullet A_2) + \cdots + (b_{K2} \bullet A_K)\right] 2^{-2} \\
& \qquad \vdots \\
& + \left[(b_{1(N-1)} \bullet A_1) + (b_{2(N-1)} \bullet A_2) + \cdots + (b_{K(N-1)} \bullet A_K)\right] 2^{-(N-1)}
\end{aligned}
$$

$$
y = -\boxed{\sum_{k=1}^{K} (b_{k0}) \bullet A_k} + \boxed{\sum_{n=1}^{N-1} \left[b_{1n} \bullet A_k + b_{2n} \bullet A_2 + \cdots + b_{Kn} \bullet A_K\right] 2^{-n}}
$$

$$
\boxed{y = -\sum_{k=1}^{K} A_k \bullet (b_{k0}) + \sum_{n=1}^{N-1} \left[\sum_{k=1}^{K} A_k \bullet b_{kn}\right] 2^{-n}} \qquad \ldots(4)
$$

How is the hardware realization ?

- Consider the equation (4) rewritten as:

$$y = \sum_{n=1}^{N-1} \left[\sum_{k=1}^{K} A_k b_{kn} \right] 2^{-n} + \sum_{k=1}^{K} A_k (-b_{k0})$$

  - $\left[\sum_{k=1}^{K} A_k b_{kn} \right]$ has only *2*^{K} possible values
  - $\sum_{k=1}^{K} A_k (-b_{k0})$ has only *2*^{K} possible values
  - With the sign bit as an input,

    we can store it in a ROM of size=*2*2*^{K}

Example

- Let number of taps *K* be 4
- The fixed coefficients are A_1 =0.72, A_2= -0.3, A_3 = 0.95, A_4 = 0.11

$$y = \sum_{n=1}^{N-1} \underbrace{\left[\sum_{k=1}^{K} A_k b_{kn} \right]} 2^{-n} + \underbrace{\sum_{k=1}^{K} A_k(-b_{k0})} \quad \ldots(4)$$

- We need $2^K = 2^4$ = 16-words ROM

ROM: Address and Contents

$$\left[\sum_{k=1}^{4} A_k b_{kn}\right] = A_1 b_{1n} + A_2 b_{2n} + A_3 b_{3n} + A_4 b_{4n}$$

b_{1n}	b_{2n}	b_{3n}	b_{4n}	Contents
0	**0**	**0**	**0**	**0**
0	**0**	**0**	**1**	**A_4=0.11**
0	**0**	**1**	**0**	**A_3=0.95**
0	**0**	**1**	**1**	**A_3+ A_4=1.06**
0	**1**	**0**	**0**	**A_2=-0.30**
0	**1**	**0**	**1**	**A_2+ A_4= -0.19**
0	**1**	**1**	**0**	**A_2+ A_3=0.65**
0	**1**	**1**	**1**	**A_2+ A_3 + A_4=0.75**
1	**0**	**0**	**0**	**A_1=0.72**
1	**0**	**0**	**1**	**A_1+ A_4=0.83**
1	**0**	**1**	**0**	**A_1+ A_3=1.67**
1	**0**	**1**	**1**	**A_1+ A_3 + A_4=1.78**
1	**1**	**0**	**0**	**A_1+ A_2=0.42**
1	**1**	**0**	**1**	**A_1+ A_2 + A_4=0.53**
1	**1**	**1**	**0**	**A_1+ A_2 + A_3=1.37**
1	**1**	**1**	**1**	**A_1+ A_2 + A_3 + A_4=1.48**

Plus and Minus Points

- The architecture has accomplished MAC without an explicit multiplier
- The size of ROM, however,grows exponentially with each added input address line
- For each element in a vector, we have an address line. So we'll have K address lines
- If K is 16, this implies 2^{16} (i.e., 64K) of ROM

Offset binary coding to reduce ROM size

$$x_k = \frac{1}{2}[x_k - (-x_k)]$$

$$x_k = -b_{k0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n}$$

2‘s-complement

$$-x_k = -\bar{b}_{k0} + \sum_{n=1}^{N-1} \bar{b}_{kn} 2^{-n} + 2^{-(N-1)}$$

$$x_k = \frac{1}{2}\left[-\left(b_{k0} - \bar{b}_{k0}\right) + \sum_{n=1}^{N-1} \left(b_{kn} - \bar{b}_{kn}\right) 2^{-n} - 2^{-(N-1)}\right]$$

Rewriting x_k differently, we have

$$x_k = \frac{1}{2}\left[-\underbrace{\left(b_{k0} - \bar{b}_{k0}\right)} + \sum_{n=1}^{N-1} \underbrace{\left(b_{kn} - \bar{b}_{kn}\right)} 2^{-n} - 2^{-(N-1)} \right]$$

- Define: Offset Code

$$c_{kn} = \left\{ \begin{array}{ll} b_{kn} - \bar{b}_{kn} & , n \neq 0 \\ -(b_{kn} - \bar{b}_{kn}) & , n = 0 \end{array} \right. \qquad where \quad c_{kn} \in \{-1, 1\}$$

- Finally

$$x_k = \frac{1}{2}\left[\sum_{n=0}^{N-1} c_{kn} 2^{-n} - 2^{-(N-1)} \right]$$

Using the new x_k we have

$$x_k = \frac{1}{2}\left[\sum_{n=0}^{N-1} c_{kn} 2^{-n} - 2^{-(N-1)}\right]$$

- Substitute the new x_k in $y = \sum_{k=1}^{K} A_k x_k$

$$y = \frac{1}{2}\sum_{k=1}^{K} A_k \left[\sum_{n=0}^{N-1} c_{kn} 2^{-n} - 2^{-(N-1)}\right]$$

$$y = \frac{1}{2}\sum_{k=1}^{K}\sum_{n=0}^{N-1} A_k c_{kn} 2^{-n} - \frac{1}{2}\sum_{k=1}^{K} A_k 2^{-(N-1)}$$

$$y = \sum_{n=0}^{N-1}\frac{1}{2}\sum_{k=1}^{K} A_k c_{kn} 2^{-n} - \frac{1}{2}\sum_{k=1}^{K} A_k 2^{-(N-1)} \quad \ldots(9)$$

The New Formulation in Offset Code

$$y = \sum_{n=0}^{N-1} \frac{1}{2} \sum_{k=1}^{K} A_k c_{kn} 2^{-n} \boxed{- \frac{1}{2} \sum_{k=1}^{K} A_k 2^{-(N-1)}}$$

If we let

$$Q(c_{1n} c_{2n} \cdots c_{Kn}) = \frac{1}{2} \sum_{k=1}^{K} A_k c_{kn}$$

and

$$\boxed{Q(0) = -\frac{1}{2} \sum_{k=1}^{K} A_k}$$

Constant

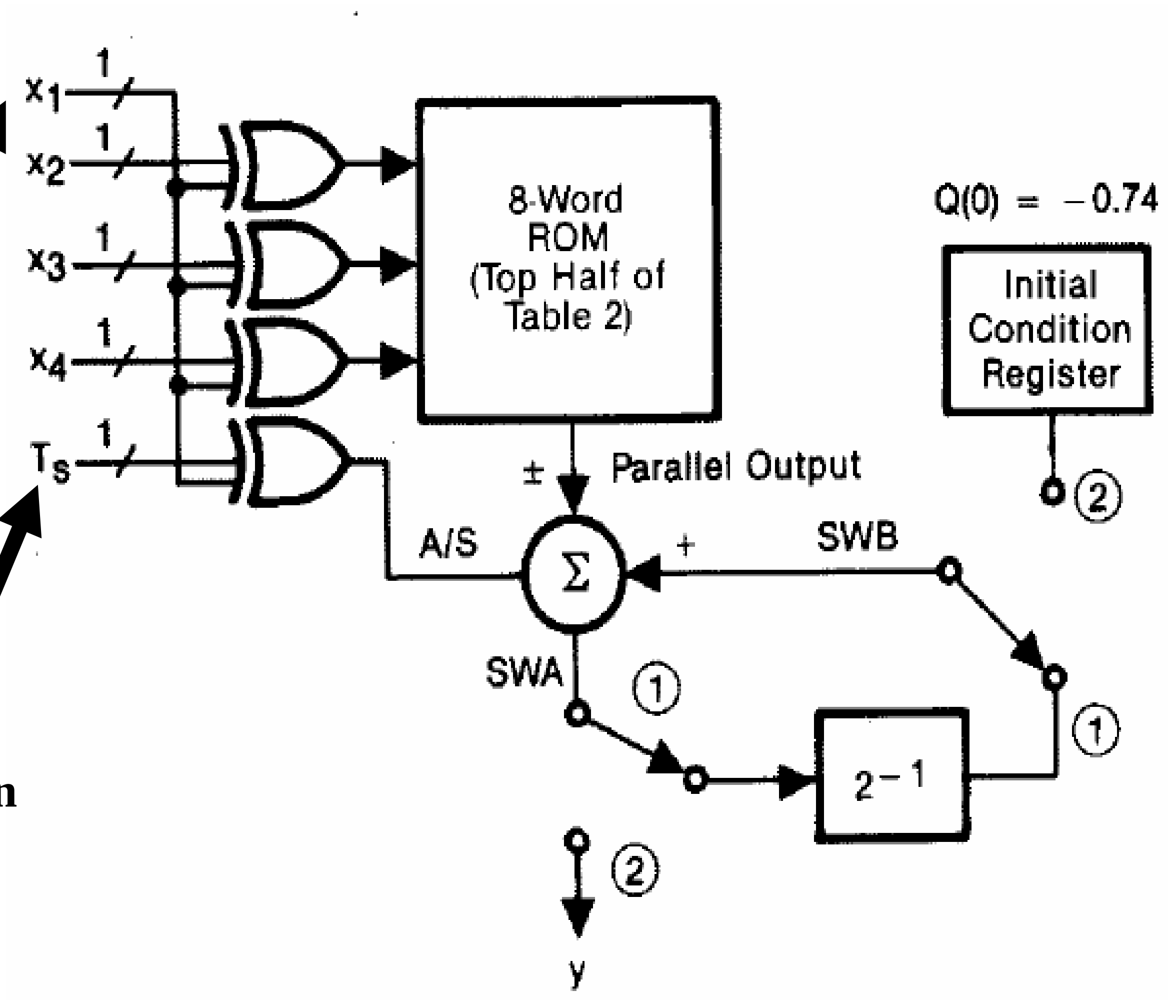
$$\boxed{y = \sum_{n=0}^{N-1} Q(c_{1n} c_{2n} \cdots c_{Kn}) 2^{-n} + 2^{-(N-1)} Q(0)}$$

The Gain: have reduced storage to 8 rows

b_{1n}	b_{2n}	b_{3n}	b_{4n}	c_{1n}	c_{2n}	c_{3n}	c_{4n}	Contents
0	0	0	0	-1	-1	-1	-1	$-1/2\ (A_1 + A_2 + A_3 + A_4) = -0.74$
0	0	0	1	-1	-1	-1	1	$-1/2\ (A_1 + A_2 + A_3 - A_4) = -\ 0.63$
0	0	1	0	-1	-1	1	-1	$-1/2\ (A_1 + A_2 - A_3 + A_4) = 0.21$
0	0	1	1	-1	-1	1	1	$-1/2\ (A_1 + A_2 - A_3 - A_4) = 0.32$
0	1	0	0	-1	1	-1	-1	$-1/2\ (A_1 - A_2 + A_3 + A_4) = -1.04$
0	1	0	1	-1	1	-1	1	$-1/2\ (A_1 - A_2 + A_3 - A_4) = -\ 0.93$
0	1	1	0	-1	1	1	-1	$-1/2\ (A_1 - A_2 - A_3 + A_4) = -\ 0.09$
0	1	1	1	-1	1	1	1	$-1/2\ (A_1 - A_2 - A_3 - A_4) = 0.02$
1	0	0	0	1	-1	-1	-1	$-1/2\ (-A_1 + A_2 + A_3 + A_4) = -0.02$
1	0	0	1	1	-1	-1	1	$-1/2\ (-A_1 + A_2 + A_3 - A_4) = 0.09$
1	0	1	0	1	-1	1	-1	$-1/2\ (-A_1 + A_2 - A_3 + A_4) = 0.93$
1	0	1	1	1	-1	1	1	$-1/2\ (-A_1 + A_2 - A_3 - A_4) = 1.04$
1	1	0	0	1	1	-1	-1	$-1/2\ (-A_1 - A_2 + A_3 + A_4) = -\ 0.32$
1	1	0	1	1	1	-1	1	$-1/2\ (-A_1 - A_2 + A_3 - A_4) = -\ 0.21$
1	1	1	0	1	1	1	-1	$-1/2\ (-A_1 - A_2 - A_3 + A_4) = 0.63$
1	1	1	1	1	1	1	1	$-1/2\ (-A_1 - A_2 - A_3 - A_4) = 0.74$

Inverse symmetry

DA Hardware with Offset Binary Coding



x_1 **selects between the two symmetric halves**

T_s **indicates when the sign bit arrives**

How to reduce ROM size further ?

One approach to reduce the ROM size is by decomposing the ROM

In particular, can divide the N address bits of the ROM can be divided into (N/K) groups of K bits

So a ROM of size 2N   can be divided into N/K ROMs of size 2K

Will need an adder to add the outputs of these ROMs and a multi-input accumulator

This is an active area of research .

Comparisons in initial years of DA

- Traditional comparisons are with multiplier-based solutions for problems pertaining to filters
- When DA was devised (in the 1970s), the comparisons given were in terms of number of TTL ICs required for mechanization of a certain type of filter
- In particular, for an eighth order digital filter operating at a word rate close to 1 MHz, 72 ICs with a total power consumption of about 30 W was stated with the DA approach while 240 ICs with a power dissipation of 96 W was indicated for a multiplier-based solution

Current Status: DA vs Mplr on FPGA

- A study of computation of Y = aX1 + bX2 + cX3 was performed with code developed in Verilog. Elements were chosen to have 8-bit size
- A Spartan-XC3S500E based synthesis was carried out in Xilinx ISE 10.1
- When built-in multipliers were used, the resource consumption was 35 slices and 3 multipliers. The combinational path delay was 15.17 ns.
- For the DA-based solution, 47 slices were used and the max frequency of operation was 142.572 MHz.
- Power consumption (obtained using XPower) for DA-based approach was slightly less than that for the multiplier-based solution

References

- A. Peled and B. Liu, A new hardware realization of digital filters, IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-22, No. 6, pp. 456-462, Dec. 1974
- S. A. White, Applications of distributed arithmetic to digital signal processing: A tutorial review," IEEE ASSP Magazine, July, 1989
- URL http://staff.kfupm.edu.sa/ITC/miali/Distributed Arithmetic.ppt
- Xilinx Application Note, The role of distributed arithmetic in FPGA-based signal processing, *www.**xilinx**.com/**appnotes**/theory1.pdf*