4.2. Parallel Synthesis of Distributed Inner Product Processors

Direct hardware implementation of large inner product operations are always difficult because of the complexity of the multiplier modules. This sub-chapter suggests a new multiplier synthesis method for this type of arithmetic operation. The fully concurrent, bit serial vector multiplication architecture is intended for FPGA or ASIC implementation. The bit level optimization algorithm exploits the inherent bit pattern coincidences of the multiplier coefficients and using the deepest descent iteration steps, generates the minimal hardware implementation of the vector multiplications. The merged, distributed arithmetic module results in more than 30% resource savings. Higher bit clock rate can be achieved due to the smaller area and reduced average fan-out. An example digital FIR filter architecture is synthesized using the algorithm and implemented on the Xilinx 4000 FPGA family.

4.2.1. The inner product operation

In many signal processing applications the calculation of the inner products of vectors are the most complicated and time/area consuming operations. Consider the inner product $\boldsymbol{y}$ of two vectors $\boldsymbol{x}$ and $\boldsymbol{h}$ given by

$$y = x^T h = \sum_{i=1}^{N} x_i h_i \qquad \textbf{(4.2.1)}$$

where $\boldsymbol{x}$ is a variable and $\boldsymbol{h}$ is an a priori known constant series. The obvious evaluation of Eq. 4.2.1 over i requires $\boldsymbol{N}$ multiplications and $\boldsymbol{N\text{-}1}$ additions. In this algorithm every product is calculated separately and then accumulated. To minimize roundoff noise effects full-precision accumulation should be applied. The modern DSP processors with internal parallel hardware multiplier/accumulator (MAC) architecture support this type of computation, what can be characterized as word serial-bit parallel computation scheme.

If $\boldsymbol{x}_i$ and $\boldsymbol{h}_i$ have $\boldsymbol{B}_x$ and $\boldsymbol{B}_h$ bit binary representations, respectively, then one obtains Eq. 4.2.2, where $\boldsymbol{x}_{i,j}$ and $\boldsymbol{h}_{i,k}$ are the j-th and k-th binary coefficient of $\boldsymbol{x}_i$ and $\boldsymbol{h}_i$.

$$y = \sum_{i=1}^{N} \sum_{j=0}^{B_x-1} \sum_{k=0}^{B_h-1} x_{i,j} h_{i,k} 2^{-(j+k)} \qquad \textbf{(4.2.2)}$$

Many alternative realizations exist. When the bits of x_i words are available simultaneously, then rewriting Eq. 4.2.2, another computation method can be derived,

$$y = \sum_{k=0}^{B_x-1} \sum_{i=1}^{N} \sum_{j=0}^{B_h-1} h_{i,j} 2^{-j} x_{i,k} 2^{-k} = \sum_{k=0}^{B_x-1} f(x_{i,k}) 2^{-k} \qquad \textbf{(4.2.3)}$$

where the function $f(x_{i,k})$ is given by

$$f(x_{i,k}) = \sum_{i=1}^{N} h_i x_{i,k} \qquad \textbf{(4.2.4)}$$

In this formula, the different values of $f(x_{i,k})$ can be precomputed, quantized to the wordlength B_x and stored in a 2^L-word size look-up table. During calculation the N-tuple $x_{i,k}$ is presented as an address to the memory, the precomputed value of $f(x_{i,k})$ outputted and added to the rightshifted earlier partial sum. This type of calculation of the inner product is called distributed arithmetic (DA), as there are no distinct products of $h_i x_i$ anymore. The concepts was introduced in [Pel74], the theoretical background and some different structures was discussed in [Bur77]. DA is very flexible and explains the trade off between memory and arithmetic. Basically it falls into the word parallel, bit serial method, so the operation needs minimum B_x computation cycle. Depending on N and B_x the look-up table based DA arrangement can be comparable in speed with the first solution.

The purpose of this paper to introduce a more deeper analysis of inner product operation down to the bit level and at the same time to give efficient computational algorithm specifically for direct hardware implementation on FPGA or ASIC devices. Until now only memory based DA method was used, because the availability of the functional units. FPGA technology now provides and at the same time, needs different methods to realize digital arithmetic. The basically gate-array like internal architecture of FPGAs consists of relatively simple, but powerful logic blocks or modules and programmable routing resources.

4.2.2 Bit level analysis of the inner product operation

There are other possible computations of Eq. 4.2.1. As it was shown in [Bur77], one dimensional coefficient vectors represent two-dimensional data structures, where the rows are the binary bit string representations of the coefficient words with an implicit + sign between the matrix row elements. In that sense the inner product operation is a special vector matrix product, where the elements of the $\boldsymbol{H}$ matrix are formed by the value set of $\boldsymbol{h}_{i,j}$, in two's complement notation $\boldsymbol{h}_{i,j} \in \{0,1\}$ Eq. 4.2.5.

$$y = x^T H = [x_1 x_2 .. x_N] \begin{bmatrix} h_{1,1}2^0 & h_{1,2}2^{-1} & . & h_{1,\boldsymbol{B}_h-1}2^{-\boldsymbol{B}_h+1} \\ h_{2,0}2^0 & h_{2,1}2^{-1} & . & h_{2,\boldsymbol{B}_h-1}2^{-\boldsymbol{B}_h+1} \\ . & . & . & . \\ . & . & . & . \\ h_{N,0}2^0 & h_{N,1}2^{-1} & . & h_{N,\boldsymbol{B}_h-1}2^{-\boldsymbol{B}_h+1} \end{bmatrix} \quad \textbf{(4.2.5)}$$

An obvious implementation for the product is the application of $\boldsymbol{N}$ serial-parallel multipliers with $\boldsymbol{B}_h$ wordlength followed by the summation of the serial product outputs [Ait91]. This hardware implementation can be built up from two types of functional units: serial adders and bit delays. Non-zero elements of the $\boldsymbol{H}$ matrix show the number of partial products (data words times one bit of coefficient) in the final result and in fact gives the figure of computational complexity of the vector product. Any reduction in the $\boldsymbol{H}$ matrix elements will decrease the complexity of the computation and of course the required hardware. The application of the Canonical Signed Digit (CSD) code [Rei60] for the a priori known $\boldsymbol{h}_i$ coefficients is the first step to decrease the complexity. Using CSD code in a $\boldsymbol{B}_h$-bit coefficient the number of non-zero digits are statistically $\boldsymbol{B}_h/3$, allowing 0, +1 and -1 weights to each digit. The hardware cost of the bit serial subtractor is the same as the adder, so signed partial product can be handled without extra hardware. This method is widely known and applied for dedicated bit serial VLSI signal processing [Den85], and an FIR filter implementation on FPGA elements was presented in [Feh92a]. In this paper a new algorithm is introduced to further reduce hardware complexity, converting the multiply and accumulate operation to a special

version of DA. The algorithm based on the distributive property of multiplication :

$$\begin{aligned}(h_{i,\alpha}+h_{i,\beta})x_n &= h_{i,\alpha}x_n+h_{i,\beta}x_n\\ h_{j,\gamma}x_n+h_{k,\gamma}x_m &= h_{j,\gamma}(x_n+x_m)\end{aligned} \qquad \textbf{(4.2.6)}$$

In Eq. 4.2.6 $\boldsymbol{h}_{i,\alpha}$, $\boldsymbol{h}_{i,\beta}$, $\boldsymbol{h}_{j,\gamma}$, $\boldsymbol{h}_{k,\gamma}$ are subpolynomials of the binary representation of $\boldsymbol{h}_i$, $\boldsymbol{h}_j$, and $\boldsymbol{h}_k$ respectively, such that $\boldsymbol{h}_i=\boldsymbol{h}_{i,\alpha}+\boldsymbol{h}_{i,\beta}$, $\boldsymbol{h}_{j,\gamma} \subseteq \boldsymbol{h}_j$, $\boldsymbol{h}_{k,\gamma} \subseteq \boldsymbol{h}_k$, and $\boldsymbol{h}_{j,\gamma}=\boldsymbol{h}_{k,\gamma}$. The first form states that multiplication of variables by coefficients are dividable to groups of partial products, while the second one states that identical partial products groups can be merged. Merged multipliers are discussed in [Die90], but only for special, matching constant values and for small $\boldsymbol{N}$. For arbitrary values and large $\boldsymbol{N}$ an implementation can be found in [Feh92a], without minimization. Using the identities of Eq.4.2.6, the new algorithm exploits coincidences of sub-polynomials in the binary representation if $\boldsymbol{h}_i$ and realizes them separately from the remaining bits.

A simple example is given in Eq. 4.2.7, where the $\boldsymbol{h}_i$ (i=1..4) values are (115, -58, 152, 122), respectively, for $\boldsymbol{B}_c$=8 bit.

$$\begin{aligned} y=[x_1 x_2 x_3 x_4]&\begin{bmatrix}1 & 0 & 0 & -1 & 0 & 1 & 0 & -1\\ 0 & -1 & 0 & 0 & 1 & 0 & -1 & 0\\ 1 & 0 & 1 & 0 & -1 & 0 & 0 & 0\\ 1 & 0 & 0 & 0 & -1 & 0 & 1 & 0\end{bmatrix}\\ =[x_1 x_2 x_3 x_4 x_5]&\begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0 & 0 & -1\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 1 & 0 & 1 & 0 & -1 & 0 & 0 & 0\\ 1 & 0 & 0 & 0 & -1 & 0 & 1 & 0\\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 1\end{bmatrix}\end{aligned} \qquad \textbf{(4.2.7)}$$

Using the aforementioned CSD code, in this case the first two lines of the $\boldsymbol{H}$ matrix contains the same {1 0 0 -1 0 1} patterns, disregarding from the different sign and position of them. With the insertion of one shift operation in order to delay $\boldsymbol{x}_1$ by one cycle (this means a multiplication by 2) and a serial presubtractor a composite signal

($\boldsymbol{x}_5=2\boldsymbol{x}_1-\boldsymbol{x}_2$) is generated and multiplied by the common bit pattern, what is the new fifth row of the modified example multiplier matrix. With the introduction of $\boldsymbol{x}_5$, the coinciding bit patterns are removed from the first two rows of $\boldsymbol{H}$. In the example, $\boldsymbol{x}_2$ is fully absorbed by $\boldsymbol{x}_5$, (the second row of $\boldsymbol{H}$ is cleared by the reduction step), so $\boldsymbol{x}_2$ can be dropped out from the signal list after the first iteration. The reduction in the total number of non-zero digits in $\boldsymbol{H}$ is 3. The gain of the iteration step is always smaller than the reduction, because of the cost of the preshift and the presubtract operational units. Digits of the fifth line of $\boldsymbol{H}$ are shifted right by one position to compensate the inherent one-bit-delay of the pipeline serial subtractor.

The result will be the factorization of the $\boldsymbol{H}$ matrix, as shown in Eq. 4.2.8. The $\boldsymbol{P}$ preaddition matrix is an identity matrix completed by a column vector. The new column vector of $\boldsymbol{P}$, what defines $\boldsymbol{x}_5$, has only two non-zero elements, generally one of them is an integer power of two, expressing some shift operation before partial merging, the other is +/- 1. Equation 4.2.9 shows another interpretation of the reduction step.

$$y=[x_1\ x_2\ x_3\ x_4]\cdot \begin{bmatrix}1&0&0&0&2\\0&1&0&0&-1\\0&0&1&0&0\\0&0&0&1&0\end{bmatrix}\begin{bmatrix}0&0&0&0&0&0&0&-1\\0&0&0&0&0&0&0&0\\1&0&1&0&-1&0&0&0\\1&0&0&0&-1&0&1&0\\0&0&1&0&0&-1&0&1\end{bmatrix} \tag{4.2.8}$$

According to that, in each iteration a new $\boldsymbol{x}^i$ signal vector, and a new $\boldsymbol{H}^i$ matrix are generated.

$$y=x^{0T}H^0=x^{0T}\boldsymbol{P}^0H^1=x^{1T}H^1 \tag{4.2.9}$$

The final result after the iteration process is given in the format defined by Eq. 4.2.10.

$$y=x^{0T}H^0=x^{0T}\prod_{m=0}^{M}\boldsymbol{P}^m\quad H^{M+1} \tag{4.2.10}$$

where M is the number of iteration steps. Iteration continues until the available gain of the step is higher than one. During the iteration the non-zero elements of H are continously eliminated and the arithmetic operations are executed in the preaddition phases. The convergence is guaranteed in finite number of step, because a trivial upper limit for M is $N*B_h/2$, the maximum number of non-zero digits in H, as determined by the CSD representation.

4.2.3 The complexity reduction algorithm

The goal of the algorithm is to find the minimum cost implementation of the inner product operation for a given B_h-bit coefficient quantization. The cost function is measured in the number of functional units needed to realize the operation. One architecture or technology dependent factor is taken into account by the algorithm during the iteration, the C cost ratio of the shift and the arithmetic operations. In case of bit serial arithmetic this means the cost ratio of a simple D flip-flop and of a pipe-line serial adder/subtractor.

The first step of the algorithm is to check the h input vector for trivial reductions. The reason of this step is the symmetry property of the impulse response function of linear phase FIR filters or the possible underflow of some small numerical values.

Normal reduction steps are executed on the best available gain. Selection is based on statistics of non-zero bit coincidences. The Sn statistics matrices contain information about the number of coinciding non-zero elements of rows of H^i, as follows: $Sn(l,m)$ is the larger of coinciding identical or contradicting bit pattern of non-zero elements from the l-th and m-th rows of H^i, where the m-th row is shifted right by n positions. The diagonals of the Sn matrices are cleared, because for practical B_h values self coincidences can be ignored. In case of the former example the first two statistics matrices are given in Eq. 4.2.11. The example H matrix is also shown for easy reference

The parameter n (number of statistics matrices) is the function of two upper bounds and is dynamically changed during iterations according to Eq. 4.2.12, where

$$n = MAX\{\frac{\lceil 0.5B_h \rceil - \max|S0|}{0.5+C} \; ; \; \frac{H^i{}_{SMNZ} - \max|S0|}{0.5+C}\} \qquad \textbf{(4.2.12)}$$

$$H=\begin{bmatrix} 1 & 0 & 0 & -1 & 0 & 1 & 0 & -1 \\ 0 & -1 & 0 & 0 & 1 & 0 & -1 & 0 \\ 1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \end{bmatrix} \tag{4.2.11}$$

$$S0=\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & -1 & -2 \\ 1 & -1 & 0 & 2 \\ 1 & -2 & 2 & 0 \end{bmatrix};\ S1=\begin{bmatrix} 0 & 2 & -2 & -2 \\ -3 & 0 & -1 & -1 \\ 1 & -1 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{bmatrix};..\ Sn$$

$\boldsymbol{H}^i_{SMNZ}$ is the second maximum number of non-zero digits in rows of $\boldsymbol{H}^i$. As the iteration proceeds, the second limit becomes dominant and following $\boldsymbol{H}^i_{SMNZ}$, n will decrease.

With the knowledge of the full bit statistics and the actual $\boldsymbol{x}^i$ signal list, the available maximum gain can be determined. The gain is equal to the number of bit coincidences minus $\boldsymbol{C}$ times the required bit delay. The maximum gain assigns two rows of $\boldsymbol{H}^i$ to be modified. If any modified row becomes full zero, then the appropriate row is removed from $\boldsymbol{H}^i$ and the old signal is cleared from the signal list. It should be noted, that only the two reduced and the new rows and columns of $\boldsymbol{Sn}$ statistics matrices must be recalculated in each step, not the full matrices. Iteration continues until there is no more bit coincidences in the $\boldsymbol{H}^{M+1}$ matrix. The final partial product sum is simply generated from the last $\boldsymbol{H}^{M+1}$ matrix preserving the different binary weights carried by them. All iteration steps are printed out to a document file with the following syntax, while rows of the $\boldsymbol{H}$ matrix are cancelled out:

(input signal 1) FUNCTION {(input signal 2)} -> (output signal)

For the given simple example the full iteration process is shown below, where in the left column are the iteration steps, and in the rigth column are the final product summation according to the binary weights.

```
X1 DFF -> X1D1                X6 DFF -> XS2
X1D1 SUB X2 -> X5             XS2 DFF -> XS3
X3 ADD X4 -> X6               XS3 ADD X7 -> XS4
X3 ADD X5 -> X7               XS4 DFF -> XS5
X5 ADD X6 -> X8               XS5 DFF -> XS6
X1 SUB X5 -> X9               XS6 DFF -> XS7
X4 SUB X8 -> X10              XS7 ADD -> XS8
                  XS8 ADD X9 -> XSUM0
```

The two graphics in Fig. 4.2.1, show the obvious implementation using serial-parallel multipliers and the reduced flow graph after the minimization. The implementation cost of the first version is 16 arithmetic and 18 shift unit, and for the optimized one the corresponding figures are 9 and 7.

This type of description is a special kind of distributed arithmetic realization of the inner product operation. It is more distributed than the usual look-up table based DA. In the look-up table based DA structure only the evaluation order of the triple sum was changed, but in the ROM memory table the coefficient values remained intact.

The new algorithm destroys these values down to the bit level, and builds up the signal flow graph. The arithmetic complexity is transformed in part to the signal flow graph structure. Although the necessary routing has lost its regularity, but the average fan-out and wire length are also reduced. These two factors give higher achievable clock frequency for both the FPGA and ASIC technologies. From the description of the reduced signal flow graph, any netlist format can be generated.

4.2.4. Inner product implementation on Xilinx FPGA

A typical signal processing function is selected as an example implementation. This is a linear phase low pass FIR filter, with order of N=48, and cut off frequency of Fc=.51. Data and coefficient word-length sizes are 16 bit. This filter was selected by the author in an other publication [Feh93], so the results are comparable. The selected target technology is the XILINX 4000 third generation FPGA family [Hun90]. In this device the complex Configurable Logic Blocks (CLBs) can directly realize the typical bit serial

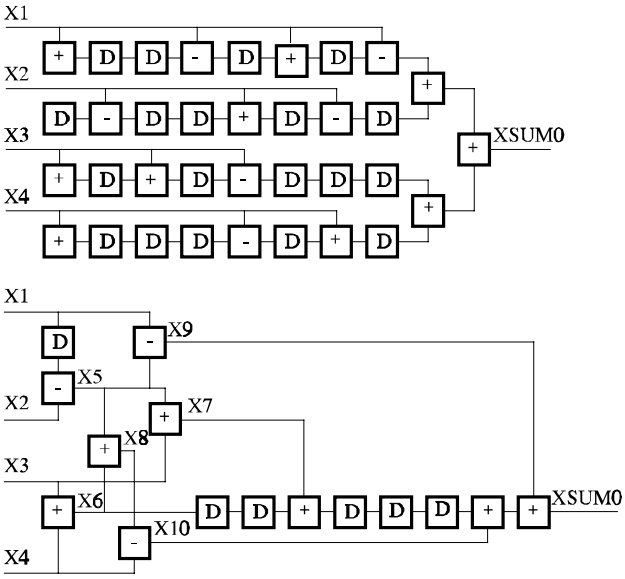

Figure 4.2.1: The original and the generated signal flow graph

functional units, together with the carry save and the pipeline flip-flop. Further, the RAM memory configuration of the CLBs provides efficient way to built up the filter delay line of size N^*B_x bit. As an extension of the general algorithm the optimazation program synthesizes the full Xilinx compatible XNF netlist of the FIR filter, including the I/O interface, delay line, control signal generator and the multiplier array. The synthesis method uses the results developed in [Feh92a, Feh93 and Feh92b]. For optimal resource utilization some functions are realized by hard macros. The example FIR filter design occupies 71% CLBs of 196 of the XC4005-5 device, and the development system timing analyzer reports maximum 27 MHz bit clock frequency without any timing driven placement and routing. Most of the CLBs are used in the inner product operation. 24 CLBs are used in the preaddition, because of the FIR filter symmetry, and 65 in the generated reduced DA flow graph. This is 33% less than the complexity of the initial $\boldsymbol{H}$ matrix after the trivial reductions. Comparing the result to the other coefficient dependent look-up table based implementation [Feh92b], where the reduction is based on the magnitude distribution of the h_i values, the new method gives about 10% achievement and better operating frequency, because parallel accumulators are not needed. The layout

of the FPGA chip is shown on Fig. 4.2.2.

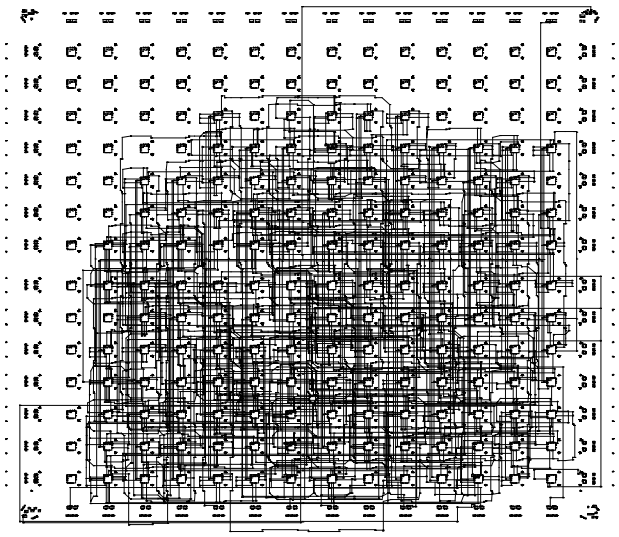

Figure 4.2.2: The layout of the example FIR filter