POFGEN: A Design Automation System for VLSI Digital Filters with Invariant Transfer Function

G. Wacey and D.R. Bull
Dept. Electrical and Electronic Engineering, University of Bristol,
Queens Building, University Walk, Bristol BS8 1TR, UK

Abstract - **This paper describes the structure, methodology and potential of a new design automation tool, POFGEN, for the generation of fixed function VLSI digital filters. The system accepts input data in the form of a coefficient vector and uses this to form a pipelined, multiplier-free architecture by employing primitive operator graph synthesis methods. The output from POFGEN is available either in the form of a structure diagram or as an HDL file for direct ASIC generation.**

I. Introduction

As a consequence of cost reductions in both computer hardware and circuit fabrication coupled with a demand for shorter design cycles, the requirement for tools which automate the IC design process has increased. This is especially noticeable in the area of real-time signal processing, where complex algorithms, requiring high computational rates, must be efficiently integrated. A number of packages have been reported for the design and compilation of application-specific DSP functions. These include FIRST [1], Cathedral [2] and FIRGEN [3]. FIRST and Cathedral I are restricted to bit-serial architectures with fixed cell libraries. More recently, increasing integration levels coupled with the quest for higher bandwidth processing has promoted the development of bit-parallel architectural synthesis tools (Cathedral II-IV and FIRGEN). These systems perform varying degrees of optimisation at the arithmetic level, with some facilitating the design of a broad range of DSP functions.

This paper introduces a new design automation system (POFGEN) which incorporates specification and architectural synthesis tools, tailored to the implementation of FIR digital filters, in both bit-serial and bit-parallel formats, for fixed function applications. The approach adopted embodies the primitive operator filter (POF) design methodology [4] in which the multiply- accumulate array is replaced by a structure based on a single directed graph employing only primitive operations (addition, subtraction and power of two multiplication). The graph is formed by encouraging the reuse of internal vertices while preserving the specified transfer function with no loss of coefficient accuracy. It offers a significant reduction in arithmetic complexity typically, for larger filters, replacing each multiplier by a single adder or subtractor.

0-7803-1254-6/93$03.00 © 1993 IEEE

II. System Overview

Figure 1 illustrates the structure of POFGEN. Filter coefficients may either be input interactively, or in the form of files generated by standard filter design packages. Multiple files representing a filter bank can also be accepted. These are combined internally to form a single file containing all unique values [5].

The transposed-form (one to many mapping) primitive operator graph is generated initially, using one of four classes of graph synthesis algorithm, characterised according to the primitive operators permitted. This is then used to form a logical graph, where precedence relationships between vertices are assigned. This allows an initial register

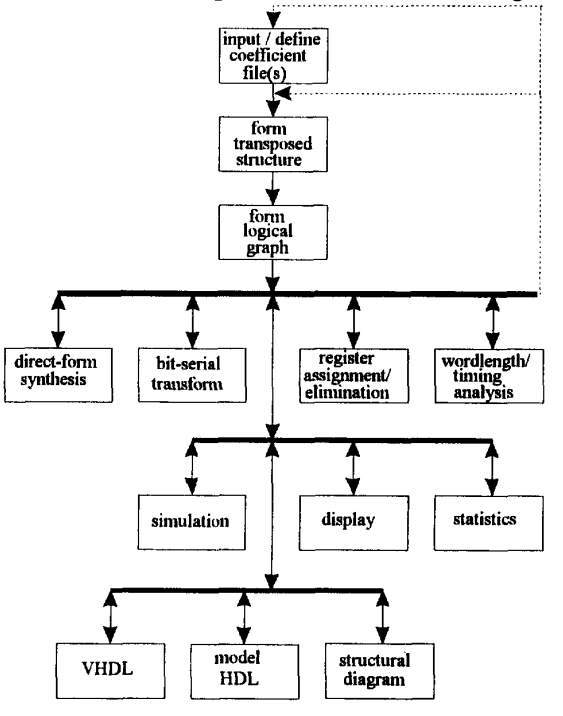


Fig.1. POFGEN structure

assignment to be performed which ensures correct synchronisation of signals at each graph vertex. Such an assignment is however generally inefficient and a process of graph reduction must be performed. At any time after logical graph formation the graph may be transposed to yield the direct form structure. Either graph type may be optimised for either bit-serial or parallel arithmetic.

Once a graph has been generated, internal data-path wordlengths can be assigned, the design simulated and a measure of circuit implementation complexity generated. The effects of signal rounding within the graph body may be assessed at this stage and simulated to give signal to quantisation noise ratios. The final pipelined architecture may be viewed in tabular or structural form and, for output purposes, the delay elements and accumulate and folding additions (the latter used in symmetrical linear phase filters) may be appended to the graph data structure to form a complete filter. Finally the filter may be output in the form of an HDL file or structure diagram.

III. Directed-Graph Synthesis Methods

A. The Primitive Operator Graph

The relationship between an input signal, x[n], an output signal, y[n], the signals at internal graph vertices, w[n], and graph output vertices, **u**[n], for the transpose form filter structure is given by the following equations:

$$\begin{aligned} \mathbf{w}[n] &= \mathbf{g}_1^T \mathbf{w}[n] + \mathbf{b}^T x[n] \\ \mathbf{u}[n] &= \mathbf{g}_2^T \mathbf{w}[n] \\ y[n] &= \sum_{i=0}^{N} u[n-i] \end{aligned} \tag{1}$$

where $\mathbf{g}_1$ and $\mathbf{g}_2$ are edge gain matrices, **b** is a row vector which maps x[n] on to $w_1[n]$ and N is the filter order.

The specified coefficient vector is used to synthesise a transposed-form graph, employing one of four available algorithms (addition only, addition/subtraction, addition/shift and addition/subtraction/shift). The most efficient algorithms for the majority of applications are the so called shift biased algorithms [4]. These attempt to minimise the number of adders/subtractors in the graph, by allowing graph edges to assume gains equal to any non-negative power of two.

Prior to algorithm execution the coefficient file is modified by dividing all elements by the maximum power of two, 2^1, which maintains an integer result. The initial graph is then synthesised using these new values and the remaining output edge gains are compensated in order to restore the original multiplier value. This pre-shifting generates a graph with fewer vertices, decreasing the internal communications overhead and reducing the average vertex data path width.

Consider an example filter with scaled integer coefficients {1,4,9,15,27,38}. The gain matrices after graph formation are as follows:

$$\mathbf{g}_1 = \begin{bmatrix} 0 & 1+2^3 & 1+2^1 & 0 & 0 & 2^4 \\ 0 & 0 & 0 & 1 & 1+2^1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{g}_2 = \begin{bmatrix} 1 & 2^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2^1 \end{bmatrix}$$

B. The Logical Graph

The edge gain matrices $\mathbf{g}_1$ and $\mathbf{g}_2$ are used to form a logical graph which provides a vehicle for the timing and pipelining of the filter. Each vertex in the graph is assigned a precedence value where the precedence level of vertex k, fed from vertices i and j, is given by:

$$\Delta_k = \max(\Delta_i, \Delta_j) + 1 \tag{2}$$

Here Δ_i and Δ_j are the precedence levels of vertices i and j respectively. The number of precedence levels present in a graph is thus $\Delta = \max(\Delta_i)$.

The precedence graph is fully pipelined by placing a padding register on every edge, at every precedence level. The delay ,d_{ij}, required on edge ij for correct data alignment and the corresponding number of padding registers, r_{ij}, needed can be computed from the edge gain matrices and are given by equations (3) and (4) respectively.

$$d_{ij} = \Delta_i - \Delta_j \tag{3}$$

$$r_{ij} = d_{ij} - 1 \tag{4}$$

C. Direct Form Transposition

Thus far only the transposed form structure has been considered. POFGEN however allows a graph to the be transposed to yield the direct-form (many to one mapping) structure. This too can be pipelined and register-reduced as described below.

Graph transposition is achieved by reversing all edge directions, exchanging branch vertices and adder vertices as appropriate and interchanging the outputs with the input. Direct transposition generally gives rise to a structure with an excessive number of precedence levels. This is due to vertices in the initial graph being frequently reused (ie having a high out-degree), resulting in a long sequence of adders in the direct form. By identifying the associated edges and combining these using a tree structure, a graph with fewer precedence levels results. If, during tree formation, the edges with the largest gains are combined first, shift elimination as described below, can be applied more effectively.

IV. REDUCTION TECHNIQUES

A. Topological Techniques

The fully pipelined graph formed as described above is generally suboptimal. It is therefore often desirable to reduce circuit complexity by eliminating any redundant padding registers. Initially duplicate paths and their associated padding registers are removed, as shown in figure 2. These occur where a vertex has a number of parallel edges emerging from it, each having associated padding registers. This overhead may be eliminated if a single shared path is used with delay values modified as follows:

$$r_{mj} = r_{nk} = 0;\ \ r_{mn} = r_{ik} - r_{ij};\ \ r_{nl} = r_{il} - r_{ik};\ \ r_{im} = r_{ij} \qquad (5)$$

Therefore $\Delta_m = \Delta j$ *and* $\Delta_n = \Delta_k$.

A second elimination technique involves translating addition/subtraction vertices to higher precedence levels as demonstrated in figure 3 where, assuming r_{jq}=0,

$$r'_{km} = 0; \qquad r'_{jq} = r_{km}; \qquad r'_{ip} = r_{ip} + r_{km};$$
$$r'_{pl} = r_{pl} - r_{km}; \qquad r'_{qn} = r_{qn} - r_{km} \qquad (6)$$

and

$$\Delta_q = \Delta_p = \Delta_k \qquad (7)$$

A final reduction technique, illustrated in figure 4 and characterised by equation (9), involves the identification of all adder/subtractor vertices with shifts on both inputs (ie for vertex k, $\{g_{ik}, g_{jk}\}>1$). The gain common to both inputs can be factorised and repositioned at the vertex output. This reduces the data path width of the vertex in a parallel system or reduces the number of shift register bits required in a serial realisation.

$$s'_{ik} = s_{ik} - \min(s_{ik}, s_{jk}); \qquad s'_{jk} = s_{jk} - \min(s_{ik}, s_{jk});$$
$$s'_{kl} = s_{kl} + \min(s_{ik}, s_{jk}) \qquad (8)$$

It should be noted that for bit-serial implementations, shifts are realised as one bit registers, and as such can double as padding registers, provided that correct vertex timing and edge gains are maintained.

B. Timing Analysis Based Techniques

The processing delay caused by each processing element is a function of the wordlength at the associated graph vertex, the delay characteristics of the cell components used and the capacitative loading due to tracking. Ignoring the effects of the latter, the delay, d_a, for a single adder comprising k+1 4-bit look-ahead-carry adder blocks is given by equation (9),

$$d_a = \max(t_4 + t_2 + (k-1)t_3, t_1) \qquad (9)$$

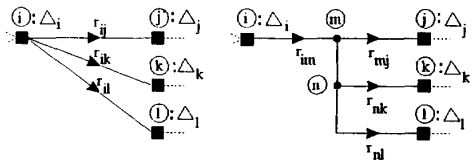
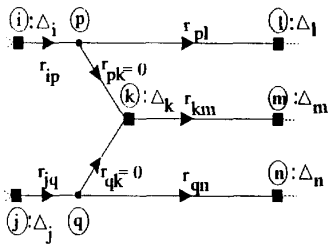


Fig.2. Duplicate path elimination



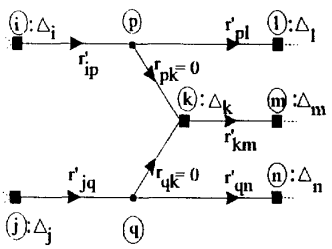


Fig.3. Duplicate path elimination II

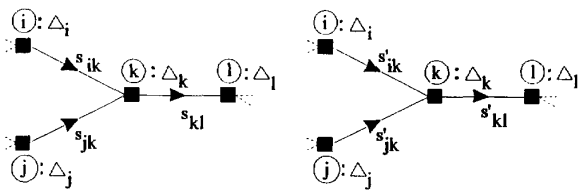


Fig.4. Shift elimination

where t_1, t_2, t_3 and t_4 represent worst case delays input to output, carry in to output, carry in to carry out and input to carry out transitions. The total delay, d_t, for a path through A consecutive adders is given by equation (10). Assuming $t_1>t_2>t_4>t_3$ and $t_1+t_3<t_2+t_4$ then,

$$d_t = At_2 + At_4 + (K-A)t_3 \qquad :K \geq A$$
$$(A-K)t_1 + Kt_2 + Kt_4 \qquad :K \leq A \qquad (10)$$

where K is the largest value of k associated with any of the A adders in the chain. Using the above equations and incorporating additional delays due to capacitative loading, each path through the graph can be optimally load balanced and any redundant pipeline registers removed.

V. Output Options

A. Structural Diagram

The fully reduced graph can be displayed in the form of a table or structural diagram. An example of the latter, for a fully pipelined version of the filter specified in section IIIa, is given in figure 5.

B. Hardware Description Language

The structure can be converted either to vendor independent VHDL [6] or to a vendor specific HDL at the gate level (currently only ES2 'model' code is supported). A series of POF parameterised parts [7] have been generated and these are used to form the HDL text file. The HDL file can take one of two forms dependent on whether a fixed or variable input wordlength is specified. The latter produces parameterised HDL file, allowing flexibility during simulation and ASICs with differing input wordlengths to be fabricated from the same design. The internal data wordlength of the filter can be assigned a maximum, critical or user defined limit. The latter is set by the designer to indicate any desired value. The maximum is determined by allowing the wordlength to grow by one bit (excluding shifts) for each adder/subtractor present in the graph. The critical case is dependent on coefficient distribution is given by equation (11),

$$B_{out} = ceil\left(\log_2\left(\sum_{i=0}^{N} |h[i]| \right) \right) + B_{in} \qquad (11)$$

where B_{in} is the input wordlength and ceil(.) returns the least integer greater or equal to its argument.

An example model code file corresponding to the structure in figure 5 is given below. This is based on the assumptions of critical case wordlength growth and 8-bit input data.

```
Include "parallellib.inc"
Part test [clk,xn(0:7),r] -> h0(0:7),h2(0:11),h3(0:11),h4(0:13),
h5(0:12)
Signal a1(0:7),a3(0:11),a4(0:9),a9(0:7)
vlpr (8) [clk,xn(0:7),r] -> a1(0:7)
pspadder (8,8,12,0,3,1) [clk,a1(0:7),a1(0:7),r] -> a3(0:11)
pspadder (8,8,10,0,1,1) [clk,a1(0:7),a1(0:7),r] -> a4(0:9)
pspadder (12,10,12,0,1,1) [clk,a3(0:11),a4(0:9),r]->h3(0:11)
pspadder (12,12,14,0,1,1) [clk,a3(0:11),a3(0:11),r]->h4(0:13)
pspadder (8,10,13,4,0,1) [clk,a9(0:7),a4(0:9),r] -> h5(0:12)
vlpr (8) [clk,a1(0:7),r] -> a9(0:7)
vlpr (8) [clk,a9(0:7),r] -> h0(0:7)
vlpr (12) [clk,a3(0:11),r] -> h2(0:11)
End
End Of File
```

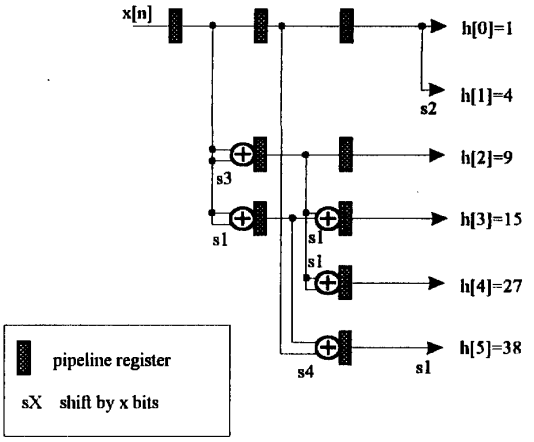


Fig.5. Example pipelined POF structure

VI. Conclusions

This paper has outlined the methodology underlying the POFGEN design automation package, together with its potential for applications requiring high throughput, fixed function FIR filters. The system, as described, is now fully operational and has been used successfully in the production of a filter bank for a 64 channel sub-band coder for video data compression. This system has now been fabricated on a single gate array. Work is continuing on the development of a full VHDL interface and on the incorporation of enhanced optimisation techniques.

Acknowledgement

The authors would like to thank Sony Broadcast and Communications, the SERC and Dave Horrocks (UWCC) for their support and assistance.

References

[1] Murray A.F. and Denyer P.B., 'A CMOS Design Strategy for Bit-Serial Signal Processing', IEEE Journal of Solid-State Circuits, Vol. SC-20, No. 3, June 1985, pp746-753,

[2] DeMan, H et al.,'Architecture Driven Synthesis Techniques for VLSI Implementation of DSP Algorithms', Proc. IEEE, Vol. 78, No. 2, Feb. 1990, pp319-335.

[3] Jain R., Yang P.T. and Yoshino T., 'FIRGEN: A Computer-Aided Design System for High Performance FIR Filter Integrated Circuits', IEEE Trans. on Signal Processing, Vol. 39, No. 7, July 1991, pp1655-1668.

[4] Bull, D.R. and Horrocks D.H., 'Primitive Operator Digital Filters', IEE Proc. Part G., June 1990, pp 401-412.

[5] Bull, D.R., Wacey, G., Stone, J.J. and Soloff, J.M., 'A Compound Primitive Operator Approach to the Realisation of Video Sub-Band Filter Banks', Proc. IEEE Intl. Conf. on Acoustics, Speech and Signal Processing, Minneapolis, USA, April 1993.

[6] 'IEEE Standard VHDL Language Reference Manual', IEEE Std 1076, Second Printing, April 1989.

[7] Wacey, G., 'A VLSI Implementation of Primitive Operator Digital Filters', MSc Dissertation, Univ. of Wales Col. of Cardiff, 1990.