Implementation of Folding Transform on 4-pole Lattice Filter

Laxya & Tarun Kumar Rawat
ECE Division, Netaji Subhas Institute of Technology, University of Delhi, New Delhi-110078
E-mail: laxya.singla@gmail.com, Tarundsp@gmail.com

Abstract - In this paper Folding transformation is applied on the 4-pole lattice filter, which is used to minimize the number of registers. In this problem the number of register will increase to 13 and when we applied the valid folding with the help of retiming. Then we will apply life time analysis and life time chart after applying life time analysis registers minimization techniques [4] are applied on 4-pole lattice filter where the number of registers required is 10. Design the required lattice filter is shown in Figure 6 so that area of the chip is minimized

Index Terms—DFG (data flow graph), Folding transformation, Retiming equation, Register minimization technique, Forward and backward allocation technique, Folded architecture.

I. INTRODUCTION

In this there have been important works on problem of reducing the number of registers. we start with the DFG where U and V are the two nodes in the DFG [1]. U and V are connected via edge e with the delay w(e) in the DFG as shown in fig1 the Folding factor is N, where in this problem N=2, The folding transformation reduced the number of functional units, and folding transformation is work on the folding set that is determined by the folding factor. The executions of the l-th iteration of the nodes U and V be scheduled at the time units Nl+u and Nl+v, respectively, where u and v are the folding orders of the Nodes U and V that satisfy 0, The functional units that execute the nodes U and V are denoted as H_u and H_v, respectively. Note that N is the number of operations folded to a single functional unit and is also referred to as the folding factor. If H_u is pipelined by P_u stages, then the result of the l-th iteration of the node U is available at the time unit Nl+u+P_u. Since the edge e has w(e) delays, the result of the l-th iteration of the node U is used by the (l+w(e))-th iteration of the node V, which is executed at N(l+w(e))+v. Therefore, the result must be stored for:

D_{f}(U \text{ to } V) \geq 0 \quad (1)

II. FOLDING TRANSFORM

For the given problem shown in fig.2 the subsets are.

S_{m1} = M_2, M_1
S_{m2} = M_3, M_4
S_{A1} = A_2, A_1
S_{A2} = A_4, A_3

The Folding Equations are:

N(w(e)-Pu+V-U \quad (2)

D_{f}(1 \rightarrow 3) = 2(0)-1+1 -0= 0 \quad (3)
D_{f}(1 \rightarrow 5) = 2(0)-1+1 -0= 0 \quad (4)
D_{f}(3 \rightarrow 7) = 2(0)-1+0 -1= -2 \quad (5)
D_{f}(4 \rightarrow 8) = 2(4)-1+1 -1= 7 \quad (6)

Fig.1  An edge U to V with w(e) delay.
Implementation of Folding Transform on 4-pole Lattice Filter

\[ D_f(4 \rightarrow 6) = 2(4) + 1 + 0 - 1 = 6 \quad (8) \]
\[ D_f(4 \rightarrow 2) = 2(4) + 1 + 1 - 0 = 8 \quad (9) \]
\[ D_f(5 \rightarrow 2) = 2(0) + 2 + 1 - 1 = -2 \quad (10) \]
\[ D_f(6 \rightarrow 1) = 2(0) + 2 + 0 - 0 = -2 \quad (11) \]
\[ D_f(7 \rightarrow 4) = 2(0) + 2 + 1 - 0 = -2 \quad (12) \]
\[ D_f(8 \rightarrow 3) = 2(0) + 2 + 1 - 0 = -2 \quad (13) \]

For the valid folding equations all these equation must be greater than ZERO, but here equation \$5, 10, 11, 12, 13\$ are not greater than zero so we have to apply retiming

### III. RETIMMING

For retiming [3] we use the retiming equations for the edges

\[ W_r(e) = w(e) + r(V) - r(U) \]
\[ r(U) - r(u) \leq D_f(U \rightarrow V)/N \]

where \( x \) is the floor of \( x \) which is the largest integer less than or equal to \( x \).

Retiming Folding constraints are:

\[ r(1) - r(3) \leq 0, r(1) - r(5) \leq 0, r(3) - r(7) \leq -1, r(3) - r(8) \leq 3, r(3) - r(4) \leq 3, r(4) - r(6) \leq 2, r(4) - r(2) \leq 4, r(5) - r(2) \leq -1, r(6) - r(1) \leq -1, r(7) - r(4) \leq -1, r(8) - r(3) \leq -1. \]

These equation can be formed by applying Floyd-Warshall Algorithm by Constraints graph and the final constrains after applying the algorithm are:

\[ r(1) = -2, r(2) = 0, r(3) = -2, r(4) = -1, r(5) = -1, r(6) = -3, r(7) = -2, r(8) = -3. \]

Now we can design retimed version of DFG shown in fig.3

\[ D_f(3 \rightarrow 8) = 2(3) + 1 + 1 - 1 = 5 \quad (18) \]
\[ D_f(3 \rightarrow 4) = 2(6) + 1 - 1 = 10 \quad (19) \]
\[ D_f(4 \rightarrow 6) = 2(1) + 1 + 0 - 0 = 1 \quad (20) \]
\[ D_f(4 \rightarrow 2) = 2(3) + 1 + 1 - 0 = 6 \quad (21) \]
\[ D_f(5 \rightarrow 2) = 2(1) + 2 + 1 - 0 = 0 \quad (22) \]
\[ D_f(6 \rightarrow 1) = 2(1) + 2 + 0 - 0 = 0 \quad (23) \]
\[ D_f(7 \rightarrow 4) = 2(1) + 2 + 0 - 0 = 0 \quad (24) \]
\[ D_f(8 \rightarrow 3) = 2(1) + 2 + 1 - 0 = 0 \quad (25) \]

Here we get valid folding so the folding sets are valid now \( D_f(U \rightarrow V) \geq 0 \). number of delay required is 13 on the place of 8.

Now we have archived the valid folding on the trade off of increase in the number of delay element as the increase in the number of delay element increases the area required so to minimizethethe number of register we implement Register minimization technique[4].

### IV. REGISTER MINIMIZATION TECHNIQUE.

This technique is required because when we implement

Folding transformation the number of register will increased for the valid folding and our aim is to reduce the number of register so we will implement register minimization technique. this can be done through by implement these steps:

- A. Life time analysis and life time chart.
- B. Forward and backward allocation technique[5].

**Life Time analysis and chart**

It is the procedure used to compute the minimum number of register required to implement a DSP algorithm. the data sample is called live when it produce the sample and though the time it is consumed. the maximum number of live variable at any time is the number of register used to implement the hardware structure. The chart of these life time analysis is shown in Table1 and life time chart is shown in figure 5.

<table>
<thead>
<tr>
<th>Node</th>
<th>Input</th>
<th>Output</th>
<th>( T_{in} \rightarrow T_{out} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>( 1 \rightarrow 1 )</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>12</td>
<td>( 2 \rightarrow 12 )</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>7</td>
<td>( 1 \rightarrow 7 )</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>3</td>
<td>( 3 \rightarrow 3 )</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>( 2 \rightarrow 2 )</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>2</td>
<td>( 2 \rightarrow 2 )</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>3</td>
<td>( 3 \rightarrow 3 )</td>
</tr>
</tbody>
</table>
Implementation of Folding Transform on 4-pole Lattice Filter

\[ T_{in} = U + P_u \]
\[ T_{out} = U + P_v + \max D_0(U \text{ to } V) \]

Fig 4: life time chart.

B. Forward and backward allocation technique

Now after the life time analysis and chart we require the maximum number of register is 8 we will implement the data allocation in these register this can be done through forward and backward register allocation. These are the steps for the forward and backward register allocation.

(i) Determine the number of register using Lifetime analysis.

(ii) Input each variable at the time step corresponding to the beginning of its lifetime. If multiple variables are input in a given cycle, these are allocated to multiple registers with preference given to the variable with the longest lifetime.

(iii) Each variable is allocated in a forward manner until it is dead or it reaches the last register. In forward allocation, if the register \( i \) holds the variable in the current cycle, then register \( i + 1 \) holds the same variable in the next cycle. If \( (i+1) \)-th register is not free then use the first available forward register.

(iv) Being periodic the allocation repeats in each iteration. So hash out the register \( R_i \) for the cycle \( l + N \), if it holds a variable during cycle \( l \).

(v) For variables that reach the last register and are still alive, they are allocated in a backward manner on a first come first serve basis.

(vi) Repeat steps 4 and 5 until the allocation is complete. The forward and backward allocation table is shown in figure 4.

V. RESULT

In this paper we had implemented all pass pole filter using folding technique, the folded lattice filter [7] is synthesized and shown in fig 6 and the retimed lattice filter is shown in fig 3, where the number of registers used is 13 and with the help of folding technique and register minimization technique [6] the number of registers is reduced to 10.

VI. CONCLUSION

We have finally presented lattice filter shown in Fig 6 with the help of folded transform applied by retiming and after the valid folded equation is achieved, where the number of register is 10 here we are able to minimize the number of registers we require 10 registers on the place of 13.

REFERENCES


