IMPLEMENTATION OF MAXIMALLY FAST LADDER WAVE DIGITAL FILTERS USING A NUMERICALLY EQUIVALENT STATE-SPACE REPRESENTATION

Oscar Gustafsson and Lars Wanhammar

Department of Electrical Engineering, Linköping University
SE-581 83 Linköping, Sweden
E-mail: {oscarg, larsw}@isy.liu.se

ABSTRACT

Ladder Wave Digital Filters is a class of IIR digital filters derived from double resistively terminated ladder networks, which gives them good stability and low sensitivity to variations in the element values. An approach to design maximally fast bit-serial ladder wave digital filters is presented based on the numerically equivalent state-space representation of the signal-flow graph. An algorithm for finding the minimal sample period from the numerically equivalent state-space representation is also given. Maximally fast filters is an efficient way to achieve low power consumption on the algorithmic level. Further, we show an approach to map the operations to an optimal hardware structure, that can be implemented using bit-parallel, digit-serial, or bit-serial arithmetic.

1. INTRODUCTION

Wave digital filters [1] is a class of IIR digital filters with highly advantageous properties. As a wave digital filter is derived from an analog reference filter, it inherits many important properties from its analog counterpart. The most interesting properties are good stability and low sensitivity with respect to variations in element values. It is also possible to use well-known design procedures to synthesize the reference filter and then transform it to the digital domain. The ladder filter structures are based on double resistively terminated ladder networks which, if correctly designed, have optimal sensitivity. The maximally fast structures are important in high-throughput systems or in low power systems after scaling the power supply voltage. In a maximally fast implementation any excess speed can be used to reduce the power consumption by reducing the power supply voltage.

2. MAXIMALLY FAST

With maximally fast we mean that the maximal sample frequency, \( f_{\text{max}} \), is equal to the upper bound on the sample frequency given by the recursive parts of the algorithm. Instead of using the maximal sampling frequency, \( f_{\text{max}} \), we often use the minimal sample period, \( T_{\text{min}} \), i.e., \( T_{\text{min}} = 1/f_{\text{max}} \).

For a given signal-flow graph we can calculate the minimal sample period [2] as

\[
T_{\text{min}} = \max \left\{ \frac{T_{\text{opt}}}{N_i} \right\}
\]

(1)

where \( T_{\text{opt}} \) is the total latency due to the operations and \( N_i \) is the number of delay elements in the directed loop \( i \). Loops that yield \( T_{\text{min}} \) are called critical loops.

The critical loops can be found by following every possible loop in the signal-flow graph and accumulating the operational latencies. If the signal-flow graph is transformed to a numerically equivalent state-space representation the latencies can be found using the algorithm presented below.

3. NUMERICALLY EQUIVALENT STATE-SPACE REPRESENTATION

For a given linear signal-flow graph with input \( x \), internal states \( v \), and output \( y \), we can write its state-space representation as

\[
\begin{bmatrix}
  v(n+1) \\
  y(n)
\end{bmatrix} = \begin{bmatrix}
  A & B \\
  C & D
\end{bmatrix} \begin{bmatrix}
  v(n) \\
  y(n)
\end{bmatrix}
\]

(2)

If the elements in \( A, B, C \) and \( D \) are computed exactly from the signal-flow graph using the truncated adaptor coefficients, the state-space representation is numerically equivalent to the original algorithm. Such an algorithm has the same robustness as the original and can be scaled optimally [3].

To find the minimal sample period of the new algorithm, the matrix \( A \) is of interest as it represents the recursive parts of the new algorithm.

3.1 Finding the Critical Path From a State-Space Representation

If we compute a new matrix, \( L \), of the same dimension as \( A \), where each element represents the latency of the multiplication of the corresponding coefficient in \( A \), the critical path can be found with the following algorithm (multiplication with 0, i.e., no connection, is handled by setting the latency smaller than zero).

1 var tmin := 0
2 function tminfromstatespace(L)
3 var usedcolumns[1:size(L)] := false;
4 for i := 1 to size(L)
5 if L[i,i] >= tmin
6 tmin := L[i,i];
7 tminiterate(L, 1, usedcolumns, 0, 0);
8 function tminiterate(L, row, usedcolumns, time, delaycount)
9 if row <= size(L)
10 for column := 1 to size(L)
11 if usedcolumns[column] = 0
12 usedcolumns[column] := true;
13 if column = row
Algorithm 1. Find $T_{\text{min}}$ for a state-space system

This algorithm searches all possible loops and have an $O(n!)$ complexity. It can however be improved with this additional test in `tminiterate` after line 9

```plaintext
var s := no_of_true_elements(usedcolumns);
var m := largest_element(L);
if s + delaycount > 0
    if ((time + (size(L) - s) * m) / (size(L) - s + delaycount) > t)
        tmin := time/delaycount;
else
    if time/delaycount > tmin
        tmin := time/delaycount;
```

The test is overestimating $T_{\text{min}}$ along the current branch. If the overestimation not will change $T_{\text{min}}$ in the worst case, there is no need to continue the search along that branch.

If the `column` variable is appended to a list which is also passed to `tminiterate` when making the call on line 16, the critical path can be found by matching the elements of the list to a sorted list containing the same elements. Elements in the same position in the lists will represent a transfer along the critical path from the node in the original list to the node in the sorted list.

### 4. IMPLEMENTATION

There are in principle two ways to achieve a maximally fast implementation of the algorithm; direct 1:m mapping of the operations in the algorithm to the hardware or transforming the algorithm using cyclic scheduling and then using a 1:1 mapping to the hardware structure [3]. Figure 1 shows the two ways to obtain a maximally fast implementation. The cyclic scheduling approach is preferred since this gives more freedom in scheduling of the operations.

![Cyclic scheduling vs. Algorithm level](image)

![1:1 mapping vs. Implementation level](image)

**Figure 1.** Mapping of an algorithm to hardware to achieve a maximally fast structure

### 4.1 Latency

The latency of an operation is defined as the time it takes to generate an output value from the corresponding input value [4]. For bit-parallel multipliers the latency is depending on the data word length, but for bit- and digit-serial it is depending on the number of fractional digits. Especially for least significant bit first bit-serial multipliers the latency is equal to the number of fractional bits.

### 4.2 Cyclic Scheduling

For a maximally fast bit-serial implementation it is generally not enough to schedule over the operations over one single sample period. This is the case if the critical loops contain more than one delay element or include operations with an execution time that is longer than the minimal sample period. If the algorithm is unfolded $m$ times the operations can be completed within the scheduling period. We are thus using cyclic scheduling for bit-serial implementations [4, 5]. The minimum number of times to unfold (the number of sample periods to schedule over) is

$$m = \left\lfloor \frac{(W_d + \max(W_f))T_{\text{clk}}}{T_{\text{min}}} \right\rfloor$$

where $W_d$ is the data word length, $W_f$ is the number of fractional bits of the coefficient and $T_{\text{clk}}$ is the clock period. The scheduling can be visualised as shown in Fig. 2.

![Cyclic scheduling visualisation](image)

**Figure 2.** Visualisation of cyclic scheduling

The maximal sampling frequency is achieved and it is determined by a critical path, $T_{\text{cp}}$, from one (or several) delay element through the arithmetic blocks $N_1, ..., N_m$ back to the delay element. Hence, $T_{\text{min}} = T_{\text{cp}}/m$.

### 4.3 Coefficient Optimization

A typical element in $A$ may look like (from the example)

$$a_{1,3} = \frac{\gamma_2(\gamma_3(\gamma_4 - 2) + \gamma_5\gamma_7)}{c_2c_3}$$

where $\gamma$ and $c_i$ are the adaptor and scaling coefficients respectively. The scaling coefficients are used to achieve maximum dynamic range space [6]. Scaling coefficients are calculated with the help of the $\ell_\infty$-norm and must be chosen as a power of two. In the example we see that at most three adaptor coefficients are multiplied, but we do also realise that cancellation of the less significant bits may occur. This means that it is not necessary that the shortest adaptor coefficients gives the lowest $T_{\text{min}}$. It also means that this may result in a lower $T_{\text{min}}$ than the original signal-flow graph.
Optimization of $T_{\min}$ can be performed by searching the coefficient space surrounding a set of valid coefficients. As $T_{\min}$ is dependent on the scaling coefficients in the filter it can however be a complex task if the $L_p$-norm has to be computed for each set of coefficients. The $L_p$-norm depends on the magnitude function, but as it does not change significantly for two slightly different coefficient sets, it is possible to use the scaling coefficients from the starting coefficient set. It is however necessary to check the $L_p$-norm after the coefficient optimization to get maximum dynamic range in the filter.

4.4 Multiple-Constant Multipliers

One way to implement the arithmetic operations, i.e., addition, subtraction, and multiplications, is to use the multiple-constant optimization described in [7]. If a value is multiplied with several constants, or in a sum-of-products, the multipliers can be significantly simplified by exploring common sub-expressions. In fact the signal-flow graph for the wave digital filter is an example of this. This reduces the hardware needed and is probably a very good solution in most cases. Multiple-constant simplifications is applicable to bit-parallel, digit-serial, and bit-serial arithmetic.

5. EXAMPLE

In the example a wave digital ladder filter with the following characteristics was synthesized

$$A_{\text{max}} = 0.7 \text{dB} \quad \omega_c T = 0.23\pi \text{ rad}$$
$$A_{\text{min}} = 45 \text{dB} \quad \omega_s T = 0.41\pi \text{ rad} \quad (5)$$

The minimum order of the filter can be found by using formulas for the standard Cauer-approximation. This gives the order to 3.7159, we choose $N = 5$ to get a large design margin.

As reference filter we choose the fifth-order cauer filter shown in Fig. 3. The filter is transformed into a wave digital ladder filter with reflection-free ports and using the method proposed in [8] to reduce the number of delay elements. The coefficients are optimized as described in section 4.3 using the coefficients given in [3] as starting point.

The filter is transformed to a wave digital filter utilizing reflection-free and using the method proposed in [8] to reduce the number of delay elements. The resulting canonic structure is shown in Fig. 4.

Optimizing the adaptor coefficients gives $T_{\min} = 7$ for at least four sets of adaptor coefficients. The set with the shortest adaptor coefficients is

$$\gamma_1 = \frac{1}{4} \quad \gamma_2 = \frac{1}{8} \quad \gamma_3 = \frac{1}{2} \quad \gamma_4 = \frac{1}{4} \quad (6)$$

These adaptor coefficients give a latency matrix

$$L = \begin{bmatrix}
7 & 5 & 9 & 8 & 10 \\
8 & 6 & 10 & 9 & 11 \\
3 & 1 & 5 & 4 & 6 \\
6 & 4 & 8 & 7 & 9 \\
4 & 2 & 6 & 5 & 7 
\end{bmatrix} \quad (7)$$

However, if $\gamma_8$ is changed to $3/8$, which is also a valid coefficient, the coefficient word length is increased and the last column in the latency matrix becomes $[x \times 4 2]^{T}$ where $x$ represents a non-connect (multiplication with zero). Hence, in this case it is better to use a longer adaptor coefficient $\gamma_8$ in order to reduce the hardware cost. The critical loops are node $v_1$ to itself, node $v_4$ to itself and node $v_1$ to node $v_4$ and back. The magnitude function of the filter is shown in Fig. 6.

Assuming a data word length of 10 bits gives $m = 3$, i.e., the scheduling of operations should be performed over three sample intervals. In Fig. 5 the scheduling formulation of the maximally fast filter is shown using ASAP-scheduling and is therefore suitable for optimization using the multiple-constant approach. The darker gray areas represents the latency of the multiplication, while the lighter gray areas represents the result. Additions are performed at the intersections marked with a bullet.
6. CONCLUSIONS

In this paper we have shown how to implement maximally fast ladder wave digital filters that achieve the theoretical upper speed bound. This is an efficient route to implement digital filters with low power consumption. The implementation is done by using a numerically equivalent state-space representation, but it is also possible to use a direct mapping of the signal flow graph. An algorithm to determine the minimal sample period of a state-space representation was also presented. The implementation allows parallel, digit-serial, and bit-serial arithmetic to be used and achieve the speed bound.

7. REFERENCES