0.1 Unfolding

It is possible to transform an algorithm to be expressed over more than one sample period. This is called unfolding and may be beneficial as it gives a higher degree of flexibility when implementing the algorithm. It may also introduce possibilities to apply other algorithmic transforms, e.g., to reduce the minimum sample period.

Consider an algorithm where we have separated the computational parts to a block, N, and delay element(s) as shown in Fig. 0.1(a). Now, in the next iteration the input and output values and states corresponds to that illustrated in Fig. 0.1(b). If we want to derive an algorithm operating over two sample periods we can assign one block, N_0, to process the even samples and one block, N_1, to process the odd samples. The input values for the blocks are shown in Fig. 0.1(c). Now, as the algorithm process two samples concurrently, a delay element will increase the signal index by two each time. Therefore, we can connect the $v(2n+2)$ output to the $v(2n)$ input with a delay element inbetween. The output $v(2n+1)$ is obviously the same as the input $v(2n+1)$. The resulting unfolded algorithm is shown in Fig. 0.1(d).

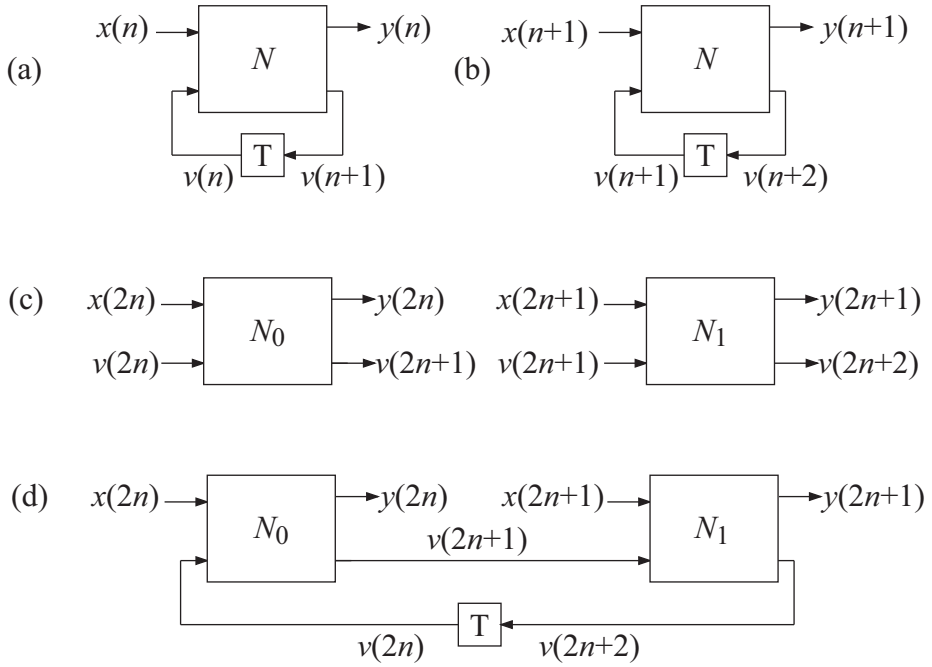

Figure 0.1. (a) An algorithm represented as the computational parts, N, and delay element(s). (b) The inputs and outputs during the next iteration. (c) The algorithm described using one block for the even samples and one for the odd samples. (d) The resulting unfolded algorithm.

Naturally, this can be generalized to other unfolding factors and more than one delay element [1]. What we are interested in is which blocks should be connected, and how many delay elements should be placed on the interconnection.

From Fig. 0.1(d) it can seen that one delay element corresponds to a connection to the next block. Similarly, two delay elements would correspond to two blocks away, etc. As the number of blocks are limited, the next block should be computed modulo, so that the first block comes after the last. Also, each time the last block is passed in the modulo count a delay element is required. The unfolding of an algorithm with a factor M can be summarized as:

- There are M computational blocks, denoted N_i, $i = 0, 1, 2, ..., M–1$.
- The input and output to block N_i are $x(Mn+i)$ and $y(Mn+i)$, respectively.
- An internal state from block N_i with L delay elements is connected to block N_j with K delay elements where

$$j = (i + L) \bmod M \tag{0.1}$$

and

$$K = \left\lfloor \frac{i + L}{M} \right\rfloor \tag{0.2}$$

The use of these equations are illustrated by an example.

EXAMPLE 0.1

Consider an algorithm that have one, two, and four delay elements at different positions of the signal flow graph. An abstracted view of the algorithm is shown in Fig. 0.2. This algorithm should be unfolded three times.

Using (0.1) and (0.2) we can derive how to connect the different blocks and how many delay elements to put on each interconnection.

$i = 0, L = 1$:

$$j = (0 + 1) \bmod 3 = 1, \; K = \left\lfloor \frac{0 + 1}{3} \right\rfloor = 0$$

$i = 0, L = 2$:

$$j = (0 + 2) \bmod 3 = 2, \; K = \left\lfloor \frac{0 + 2}{3} \right\rfloor = 0$$

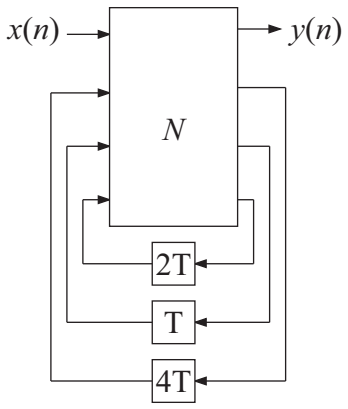

Figure 0.2. Extracted view of the algorithm used in Example 0.1.

$i = 0, L = 4$:

$$j = (0 + 4) \bmod 3 = 1, \; K = \left\lfloor \frac{0 + 4}{3} \right\rfloor = 1$$

$i = 1, L = 1$:

$$j = (1+1) \bmod 3 = 2, \; K = \left\lfloor \frac{1+1}{3} \right\rfloor = 0$$

$i = 1, L = 2$:

$$j = (1+2) \bmod 3 = 0, \; K = \left\lfloor \frac{1+2}{3} \right\rfloor = 1$$

$i = 1, L = 4$:

$$j = (1+4) \bmod 3 = 2, \; K = \left\lfloor \frac{1+4}{3} \right\rfloor = 1$$

$i = 2, L = 1$:

$$j = (2+1) \bmod 3 = 0, \; K = \left\lfloor \frac{2+1}{3} \right\rfloor = 1$$

$i = 2, L = 2$:

$$j = (2+2) \bmod 3 = 1, \; K = \left\lfloor \frac{2+2}{3} \right\rfloor = 1$$

$i = 2, L = 4$:

$$j = (2+4) \bmod 3 = 0, \; K = \left\lfloor \frac{2+4}{3} \right\rfloor = 2$$

Based on these computations it is straightforward to interconnect the three computational blocks resulting in the signal flow graph shown in Fig. 0.3.

Some observations can be made based on the example and the equations in (0.1) and (0.2). First, if L is an integer multiple of M we will always have $j = i$. Therefore, unfolding an algorithm which is expressed in z^M M times, results in M separate algorithms expressed in z, each independently processing every Mth sample. Second, the values K and j are the quotient and remainder, performing an integer division $(i + L)/M$, i.e.,

$$MK + j = i + L \tag{0.3}$$

Often, it is more convenient to just extract the input and output nodes of the delay elements in the algorithm, instead of using the abstract computational block as in Fig. 0.1. How this can be done is described in the following example.

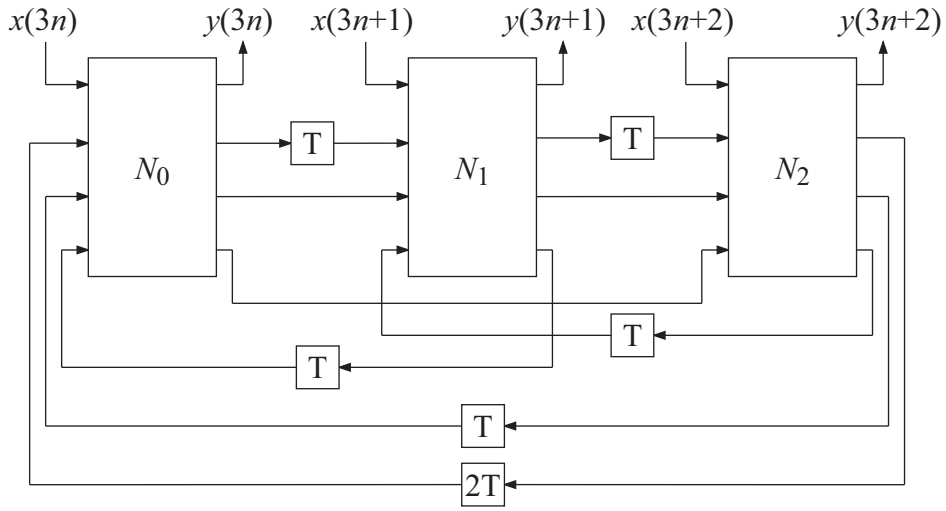

Figure 0.3. Resulting unfolded signal flow graph in Example 0.1.

EXAMPLE 0.2

Unfold a first-order allpass section based on symmetric two-port adaptors, as shown in Fig. 0.4, two times. We extract the delay element by marking the input and output nodes of the delay element. This is shown in Fig. 0.4 where a dot is used to mark the nodes v_1 and v_2.

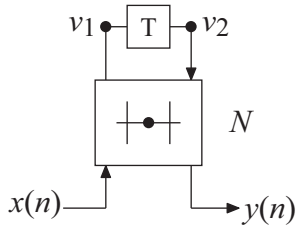

Figure 0.4. First-order WDF allpass section where the dots indicates the boundary between the computational parts and the delay element.

In the first step of the unfolding process, we place the two computational blocks as shown in Fig. 0.5. The next step is to compute the interconnection of the delay elements.

Here, we have $M = 2$ which gives

$i = 0, L = 1$:

$$j = (0 + 1)\text{mod}2 = 1\,,\; K = \left\lfloor \frac{0 + 1}{2} \right\rfloor = 0$$

$i = 1, L = 1$:

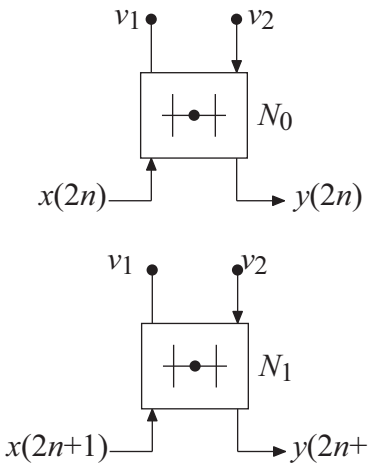

Figure 0.5. The two computational blocks obtained during unfolding of the first-order WDF allpass section in Fig. 0.4.

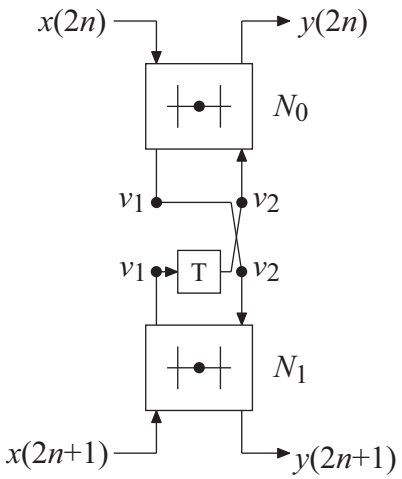

Figure 0.6. First-order WDF allpass section in Fig. 0.4 unfolded two times.

$$j = (1+1)\text{mod}2 = 0\,,\; K = \left\lfloor \frac{1+1}{2} \right\rfloor = 1$$

Hence, this gives that node v_1 of block N_0 should be directly connected to node v_2 of block N_1. Furthermore, node v_1 of block N_1 should be connected to node v_2 of block N_0 using one delay element.

The resulting algorithm is shown in Fig. 0.6. Note that the top adaptor is flipped upside down to simplify the layout of the figure.

Unfolding also finds applications when deriving single rate realization of multi-rate algorithms. For example, consider the multi-rate algorithm in Fig. 0.6. This algorithm consists of an expander followed by two filters, $H(z)$ and $G(z)$. $H(z)$ can be polyphase decomposed as

$$H(z) = H_0(z^2) + z^{-1}H_1(z^2) \tag{0.4}$$

However, this is not possible for $G(z)$. From the Noble identities we realize that we can move $H(z)$ to the lower sample rate side. This is advantageous since we avoid processing the zeros from the expander. Now, we obtain an algorithm as shown in Fig. 0.7. However, the computation of $H(z)$ and $G(z)$ are now performed at different sample rates. Instead we would like to have a realization of $G(z)$ such that two consecutive samples are processed concurrently, i.e., we would like to unfold $G(z)$ with a factor of two. The resulting single-rate algorithm is shown in Fig. 0.8.

Note that the unfolding of $G(z)$ does not directly change the computational properties. However, as all operations are performed at the same sample rate, mapping

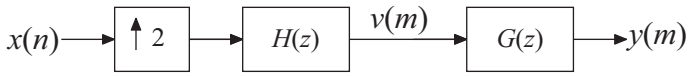

Figure 0.6. A simple multi-rate algorithm.

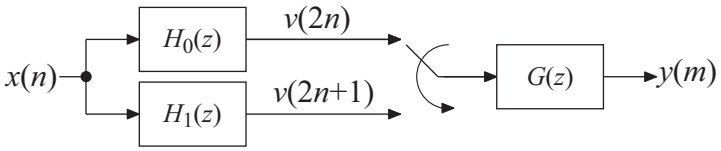

Figure 0.7. The multi-rate algorithm in Fig. 0.6 after applying the Noble identity to $H(z)$.

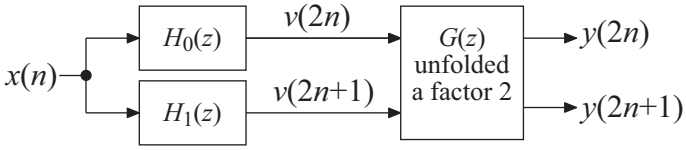

Figure 0.8. Single-rate realization of the algorithm in Fig. 0.6.

to hardware is simpler.

0.2 References

[1] K.K. Parhi, "A Systematic Approach for Design of Digit-Serial Signal Processing Architectures," *IEEE Trans. on Circuits and Systems*, Vol. 38, No. 4, April 1991, pp. 358–375.