S-38.220
Postgraduate course on signal processing in communications
FALL-99

Markus Nentwig
markus.nentwig@nokia.com

November 1999

Contents

1 Abstract

Unfolding is a technique that transforms an algorithm represented as a data flow graph into another data flow graph that performs the same algorithm, but computes several iterations of the algorithm at one clock step.
Unfolding can unravel hidden concurrencies in the DFG, which prevent that the DFG is operated at its fundamental clock period minimum given by the *iteration bound*.
With unfolding, it is always possible to achieve the iteration bound.

2 Fundamentals

A given data flow graph (DFG) processes one iteration of the underlying algorithm in each time step. It takes one set of input states and produces one set of output states.
An example is shown in figure 1.

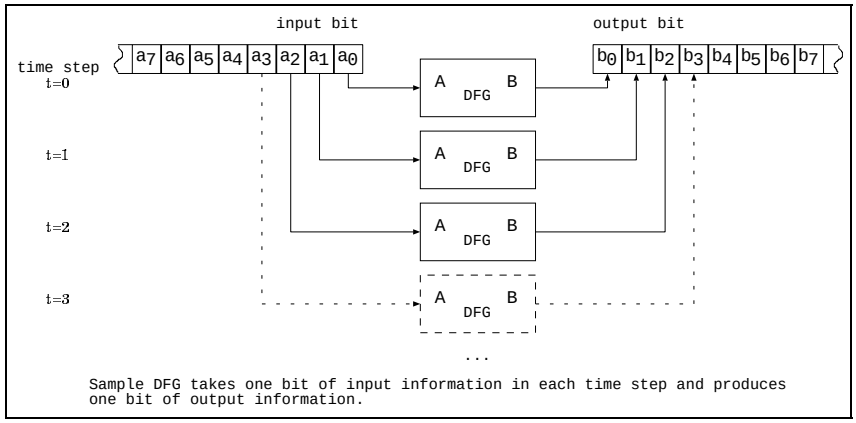

Figure 1: Original DFG

Unfolding is a technique that transforms the DFG into another DFG that exhibits the same functionality, but takes J sets of input states at one time and produces J sets of output states (example: figure 2). J is called the *Unfolding Factor*.

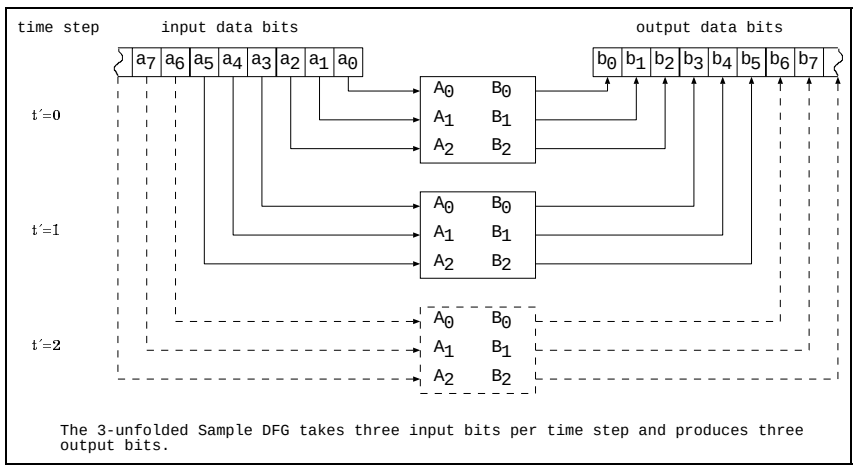

Figure 2: 3-unfolded DFG

To maintain the same input and output data rate between original and

J-unfolded DFG, the clock rate has to be reduced by the factor J.

3 Reasons for unfolding

3.1 Achieving the iteration bound

The minimum clock cycle time with which a DFG containing recursive loops can be operated is determined by the longest computation time needed by any path connecting two latches (delays), which is called *critical path*. In a recursive DFG, there is a fundamental upper limit for the clock speed. It is set by the feedback loop, which shows the highest quotient between overall computation time and number of delays.

$$T_\infty = \frac{\sum (\text{node computation times})}{\sum (\text{delays})} \tag{1}$$

This loop is called the *critical loop*.
From equation (1), it can be seen that the iteration bound T_∞ is the *average* node computation time per delay of the critical loop.

The maximum sample rate of a DFG equals T_∞, when the node computation times in the critical loop are distributed equally between the delays.

This result is obtained by simple reasoning:

- The longest computation time sum between two delays determines the critical path, which limits the maximum sample rate. If one path between two delays needs more computation time than T_∞, the iteration bound cannot be achieved, because the clock period has to be longer than T_∞.
- If a computation time sum between two delays is *smaller* than T_∞, there must be at least one other path with a higher computation time, since the average computation time of all paths is T_∞

Figure 3 shows an example of a DFG, where an equal distribution of computation times between the delays has been achieved.
Usually *retiming* can be used to redistribute the delays more evenly and reduce critical path length.

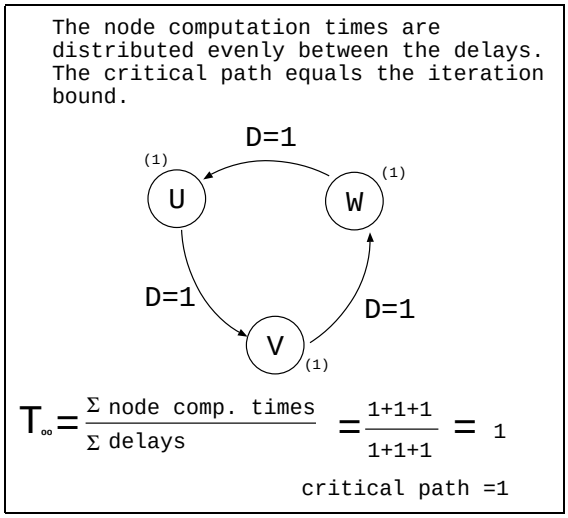

Figure 3: Equal distribution of node computation times

However, in many cases it is not possible to achieve T_∞ without unfolding the DFG.
One example for this kind of situation is shown in Figure 4.
In practical applications, this may be the general case.
Generally, computing the optimum unfolding factor is not a trivial task.

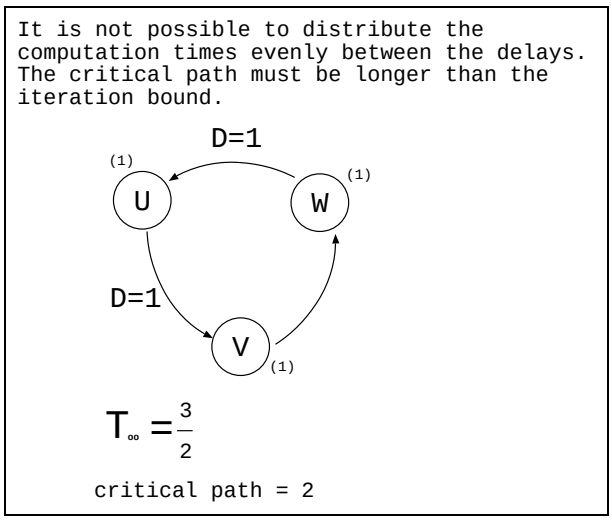

Figure 4: Computation times cannot be distributed equally

It can be found by consecutively unfolding the DFG with increasing unfolding factor and calculating the iteration bound, starting with $J = 2$.
This method may be limited by the available computational resources.

An upper bound for the optimum unfolding factor can be found as the least common multiple of the number of delays in any possible loop. However, with increasing complexity of the DFG, the resulting numbers are too high to be practical.
A further discussion on finding the optimum unfolding factor is given in [3].

A special case occurs when a single node computation time takes longer than T_∞ and cannot be split up into several nodes. It is clearly not possible to achieve T_∞ without unfolding.
Figure 5 shows an example.

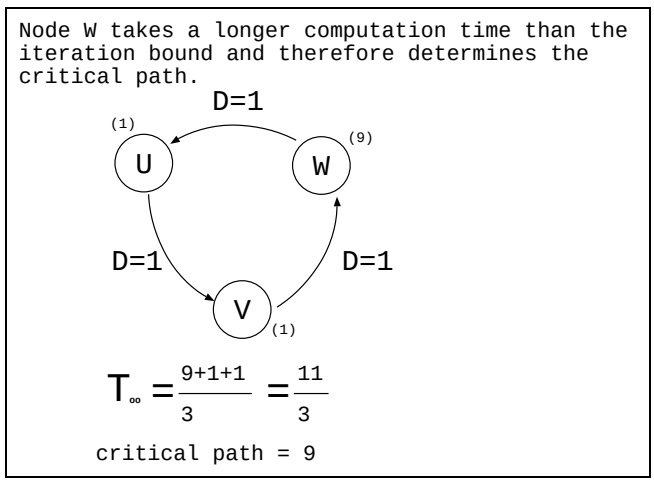

Figure 5: Node computation time T_∞

In this special case $\lceil \frac{t_U}{T_\infty} \rceil$-unfolding should be used, where $\lceil x \rceil$ is the smallest integer number greater or equal to x.
In general, unfolding a DFG with a sufficiently high unfolding factor results in a *perfect-rate* DFG. In a perfect-rate DFG, each loop contains only one delay.
Since having only one delay per loop implies an "equal" distribution of the computation times between delays, a perfect-rate DFG will always achieve the iteration bound.

3.2 Increasing the throughput of non-recursive algorithms

A non-recursive algorithm does not contain any feedback loops. This results in a fundamental difference in comparison to a recursive algorithm: A non-

recursive algorithm has no iteration bound and therefore no fundamental limit on data throughput.
By unfolding the DFG with a sufficiently high unfolding factor, arbitrary high throughput can be achieved.

3.3 Power/area trade-off in non-recursive algorithms

Reducing the supply voltage will increase the computation time of the nodes and lower the maximum permittable clock frequency. To regain the data throughput of the original DFG, the DFG can be unfolded into several parallel versions. Power consumption is reduced at the cost of more chip area.

3.4 Power savings in recursive algorithms

Unfolding increases the number of nodes by the factor of J. The number of delays remains the same, except when the DFG makes use of switches/multiplexers. Therefore, the unfolded circuit needs approximately J times the chip area of the original circuit.
An unfolded circuit executes J iterations of the underlying algorithm in one clock period. Since the unfolded DFG describes the same problem as the original DFG, the number of gate input transitions per input bit does not change.
The power consumption of a CMOS circuits is mainly caused by switching action, so unfolding alone should have no substantial effect on power consumption.
The clock rate decreases by the factor of J. The lower clock rate may allow to lower the supply voltage when the unfolded algorithm approaches its iteration bound closer than the original one.
The iteration bound increases by the factor of J, too ([1], p. 126). The iteration bound is *"the lower bound on the iteration or sample period of the DSP program regardless of the amount of computing resources available"*([1], p. 46).
So in recursive algorithms, a reduction in power consumption can be achieved by finding an implementation that approaches the iteration bound more closely.
If the iteration bound has already been achieved without unfolding and the algorithm is operating at its maximum permittable frequency, lowering the supply voltage may not be possible, because it would increase the

computation time of the critical path.

3.5 Flexible Architecture DSPs

Some modern DSPs can operate in a mode that allows a high-precision (for example 32-bit) DSP to be treated as several low precision DSPs (for example four times eight-bit). Unfolding can be used to implement an algorithm in a parallel form suitable for those DSPs [4].

3.6 Transforming serial to parallel architectures

The data stream going into and out of the DFG may represent numbers. Usually several bits are grouped together then to form a *word*.
There are several ways of implementing the algorithm:

- Bit-serial processing
  The algorithm takes one bit of each input word at each clock cycle.
- Bit-parallel processing
  The algorithm takes one word at each clock cycle.
- Digit-serial processing
  The word is divided into several groups of bits forming *digits*. The algorithm takes one digit of each input word at each clock cycle.

Unfolding can be used to transform a bit-serial algorithm into a bit-parallel or a digit-serial algorithm.

4 Properties of unfolding

4.1 Constant number of delays

Unfolding a DFG that does not contain any switches results in a DFG that contains the same number of delays.

4.2 Iteration bound

Unfolding a DFG with the iteration bound T_∞ results in a J-unfolded DFG with the iteration bound $J \cdot T_\infty$.
If the original DFG achieves the iteration bound, no increase in throughput is gained by unfolding:

- The original DFG processes one set of input data per clock period T_∞ and produces one set of output data.
- The J-unfolded DFG processes J sets of input data per clock period $J \cdot T_\infty$.

The throughput of both DFGs is the same.

4.3 Retiming and unfolding

Any clock cycle period that can be achieved by first retiming and then unfolding a given DFG can also be achieved by first unfolding and then retiming the DFG.

5 Unfolding Algorithm

A DFG can be unfolded using a simple algorithm:

- For each node U, draw J nodes $U_0, U_1, ..., U_{J-1}$
- For each edge $U \rightarrow V$, draw J edges from the node U_a to V_b with c delays for $0 \leq a \leq J-1$.
  The numbers b and c are computed according to the following equations, where w is the number of delays in the edge of the original DFG.

$$b = (a + w)\%J \tag{2}$$

$$c = \left\lfloor \frac{a+w}{J} \right\rfloor \tag{3}$$

  $\lfloor x \rfloor$ is the floor of x, which is the smallest integer number less than or equal to x.
  $u\%v$ is the remainder after dividing u by v.

Modifications have been proposed on the algorithm with the intent of reducing the computational effort for unfolding [2].

6 Unfolding switches / multiplexers

Some algorithms operate on bit streams that represent numerical data. Usually, several bits are grouped together to represent one *word*. If the algorithm contains feedback loops, it usually has to be set back to a defined state for each new word. This can be done using *switches*, which correspond to *multiplexers* in a physical circuit.
An example for this kind of algorithm is given in figure 6. A bit-serial addition is implemented by a recursive algorithm, which has to be reset for each new word.

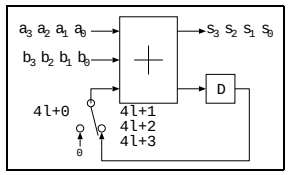

Figure 6: Bit-serial adder

6.1 Unfolding an edge with a switch

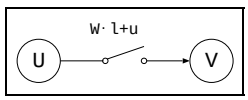

Figure 7: A switch

The edge $U \rightarrow V$ in figure 7 contains a switch with the switching equation $W\,l + u$.
It is assumed that the unfolding factor J is a multiple of the switch frequency W and that the edge containing the switch does not contain any delays.
The edge unfolds into one edge connecting the node U_p to V_p with a switch with the switching equation $W_u\,l + u_u$. Its coefficients W_u, u_u and p calculate as follows:

$$W_u = \frac{W}{J} \tag{4}$$

$$u_u = \left\lfloor \frac{u}{J} \right\rfloor \tag{5}$$

$$p = u\%J \tag{6}$$

p corresponds to the vertical position of the switch in figure 9.

6.2 Example: unfolding of single switch

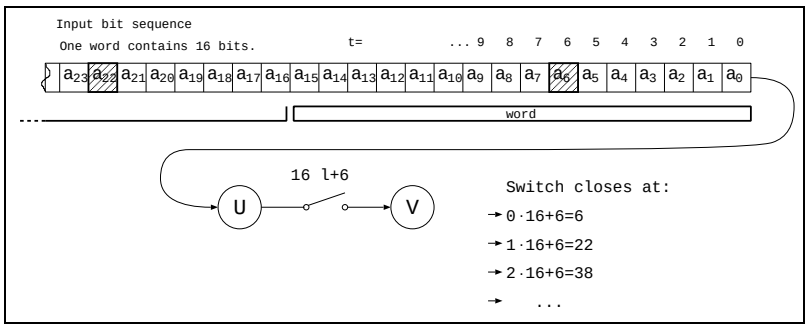

Figure 8: Algorithm containing switch

Figure 8 shows an example of a bit-serial DFG which contains a switch. It 'picks out' the 7th bit of each word (In a practical application it appears to make no sense to use one switch alone, because in this case the state of the output node will be undefined when the switch is open).
The equation of the switch in figure 8 is $W\,l + u = 16\,l + 6$.
This DFG is now to be unfolded by the unfolding factor $J = 4$.
The unfolded switch is operated by the equation $W'\,l' + u'$. It connects input node U_p to output node p.
The coefficients W', u' and p are calculated according to equations (4), (5) and (6):

$$W' = \frac{W}{J} = \frac{16}{4} = 4 \tag{7}$$

$$u' = \left\lfloor \frac{u}{J} \right\rfloor = \left\lfloor \frac{6}{4} \right\rfloor = 1 \tag{8}$$

$$p = u\%J = 6\%4 = 2 \tag{9}$$

Figure 9 shows the 4-unfolded version of the algorithm.

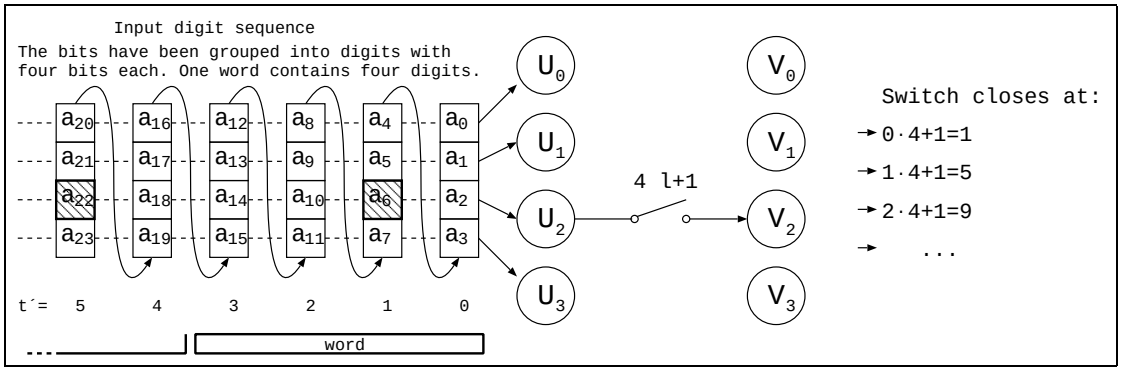

Figure 9: 4-Unfolded version of algorithm in figure 8

7 Conclusions

The concept and properties of unfolding have been discussed and an algorithm has been shown.
For non-recursive algorithms, unfolding allows to increase throughput arbitrarily. Power consumption can easily be traded off against chip area.
For recursive algorithms, unfolding allows always to reach the iteration bound and implement a given algorithm in the most efficient way. If the original DFG did not already approach the iteration bound, this improvement may increase throughput or lower power consumption.
However, in practical applications it may not always be possible to completely achieve the iteration bound because the number of parallel implementations of the algorithm grows unpractically large.

References

[1] Keshab K. Parhi: *VLSI Digital Signal Processing Systems* Wiley, 1999

[2] J.Kim: *Efficient unfolding procedure for DSP applications*
Electronics letters, 8/98

[3] Lih-Gwo Jeng Liang Gee Chen: *Rate-optimal Synthesis by Pipeline and Minimum Unfolding*
6th International Conference on VLSI Design - January 1993

[4] M. Aggarwal, N. Shanghag, N. Ahuja: *Improving the throughput of flexible-precision DSPs via algorithmic transformation*