We will start at 13:45

IMPLEMENTATION OF DIGITAL SIGNAL PROCESSING (IDSP):

ALGORITHM TRANSFORMATIONS

Sabih H. Gerez
University of Twente
Faculty of EEMCS
Computer Architecture for Embedded Systems (EWI-CAES)

COURSE PROGRESS

<table>
<thead>
<tr>
<th>Date</th>
<th>Task Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>February 19, 2021</td>
<td>Fixed-point design [How08]</td>
<td>Fixed-Point Design</td>
</tr>
<tr>
<td>February 19, 2021</td>
<td>The Arx RTL Language and Toolset [How07]</td>
<td>Arx Version with audio</td>
</tr>
<tr>
<td>February 26, 2021</td>
<td>Algorithm transformations [Par95]</td>
<td>No lecture, holiday week</td>
</tr>
<tr>
<td>March 5, 2021</td>
<td>The CORDIC Algorithm [Aza08] and [Loo09]</td>
<td>Transformation Addendum</td>
</tr>
<tr>
<td>March 12, 2021</td>
<td>Polyphase implementation of multirate filters [Law02] and [Vai09]</td>
<td>CORDIC</td>
</tr>
<tr>
<td>March 12, 2021</td>
<td>Multiplierless filter design [Hew06], [Vor07], [Aks14] and [Kot03]</td>
<td>Multiplierless Filter Design</td>
</tr>
</tbody>
</table>

PROJECT PROGRESS POLL

- Please respond on your progress on Project MAP:
  - A: I did not start at all;
  - B: I have worked on the first “pen and paper” exercises;
  - C: I have also started with the exercises on the xoc2 server;
  - D: I am (almost) ready.

FEEDBACK ON PROJECT WORK

- No intermediate grading of project reports.
- You hand in all reports simultaneously at the end of the quarter.
- If you are uncertain or stuck about some issue, you can contact me. Depending on the nature of your question I may give you some hints or just tell you to think a bit more for yourself.
TOPOICS

* Pipelining
* Retiming
* Parallel processing
* Loop unrolling
* Unfolding
* Look-ahead transformation

SPEED-UP TECHNIQUES: PIPELINING

Insert delay elements on all edges that are cut by a *cut line* through an edge of the critical path in the DFG.

* Works for acyclic DFGs.
* Schedule becomes overlapped.

Example

OPTIMAL RETIMING

* It is possible to compute the optimal positions of the delay elements in an efficient way.
* The optimization goal is to minimize the longest path from any delay element to any other. In other words, to minimize the iteration period of a non-overlapping schedule.


**RETIMING: LEISERSON ET AL.**

**CORRELATOR EXAMPLE**

Given: \( \delta(+) = 7 \) and \( \delta(?) = 3 \); \( T_{0_{\text{min}}} = ? \)

**OPTIMAL RETIMING VS. FASTEST SCHEDULE (1)**

\( T_{0_{\text{min}}} = 13 \) for non-overlapped schedule when \( \delta(+) = 7 \) and \( \delta(?) = 3 \); however, \( T_{0_{\text{min}}} = 10 \) for an overlapped schedule.

**OPTIMAL RETIMING VS. FASTEST SCHEDULE (2)**

Overlapped schedule with \( T_{0_{\text{min}}} = 10 \).

**SOME REMARKS ON \( T_{0_{\text{min}}} \)**

* Retiming does not affect \( T_{0_{\text{min}}} \) for overlapped scheduling of IDFG’s.
* The \( T_{0} \) for nonoverlapped scheduling obtained after optimal retiming may still be larger than \( T_{0_{\text{min}}} \). This is not true when all computational delays are equal to unity.
* \( T_{0_{\text{min}}} \) has been defined as an integer; a fractional \( T_{0_{\text{min}}} \) makes sense when unfolding is applied (unfolding creates a new DFG of multiple copies of the original one; see later).

---

**SPEED-UP TECHNIQUES: PARALLEL PROCESSING**

- Works for acyclic IDFGs.
- Duplicate the IDFG as often as desired speed-up factor.
- Allows any arbitrary speed-up, but is proportionally expensive.

For an IDFG, loop unrolling by a factor $n$ amounts to converting it into an acyclic graph (by cutting the delay nodes) and concatenating $n$ copies of the acyclic graph.

**LOOP UNROLLING**

Loop unrolling is the process of explicitly describing multiple iterations of some loop in order to create more parallelism.

Original loop:
```plaintext```
for (i=0; i<n; i++) {
    a[i] = f(x[i]);
    y[i] = g(a[i]);
}
```

After unrolling with a factor 2:
```plaintext```
for (i=0; i<n/2; i++) {
    a[2*i] = f(x[2*i]);
    y[2*i] = g(a[2*i]);
    a[2*i+1] = f(x[2*i+1]);
    y[2*i+1] = g(a[2*i+1]);
}
```

**UNFOLDING (1)**

- A technique for the duplication of cyclic IDFGs in combination with processing multiple inputs at a time. Cycles in the graph are preserved as opposed to loop unrolling.
- Consider the following IDFG:

**UNFOLDING (1A)**

- Topology-preserving transformation:
  - Duplicate graph
  - Remove delay elements

For an IDFG, loop unrolling by a factor $n$ amounts to converting it into an acyclic graph (by cutting the delay nodes) and concatenating $n$ copies of the acyclic graph.
**UNFOLDING (1B)**

- Add missing edges either with or without delay node.

\[ s[2k] = i[2k] + o[2k - 1] \]
\[ s[2k + 1] = i[2k + 1] + o[2k] \]
\[ o[2k] = as[2k - 1] \]
\[ o[2k + 1] = as[2k] \]

**UNFOLDING (2)**

* The precise unfolding algorithm will not be given here; it amounts to duplicating all vertices in the IDFG such that \( n \) copies of each vertex is created (\( n \) is the unfolding factor) and then to connecting these vertices with edges having an appropriate number of delay elements. The unfolded graph can also be reconstructed from the equations.

\[ s[2k] = i[2k] + o[2k - 1] \]
\[ s[2k + 1] = i[2k + 1] + o[2k] \]
\[ o[2k] = as[2k - 1] \]
\[ o[2k + 1] = as[2k] \]

**UNFOLDING (3)**

* The example IDFG after unfolding:

- Note that the unfolded IDFG has two loops with one delay element each and a computational duration of 3. Because a delay element creates an offset of two indices (2 inputs are processed in each iteration), the effective iteration period bound is equal to \( T_{0\text{min}} = \frac{3}{2} \).

**LOOK-AHEAD TRANSFORMATION (1)**

* Consider the following computation:

\[ x[n] = ax[n - 1] + u[n] \]

* It has one multiplication and one addition in the critical loop with one delay element. If \( \delta(+) = 1 \) and \( \delta(*) = 2 \), \( T_{0\text{min}} = \frac{3}{1} = 3 \).
**LOOK-AHEAD TRANSFORMATION (2)**

* Apply look-ahead transformation (think of the principle of look-ahead addition):

\[ x[n] = a(ax[n - 2] + u(n - 1)) + u[n] \]

\[ x[n] = a^2x[n - 2] + au[n - 1] + u[n] \]

* The new equation has one multiplication and one addition in the critical loop with two delays leading to \( T_{\text{min}} = \frac{3}{2} = 2 \).

* The transformation can affect the original computation (finite word length effects).

**RELATION WITH RTL SYNTHESIS**

* Multicycle operations are not so common in RTL synthesis (one normally defines a clock period for the registers and all combinational logic should execute in this period).

* RTL synthesis programs such as the Synopsys Design Compiler do support multicycle operations, by the way.

* Presented theory becomes less interesting when all computations have a unit delay:
  + Non-overlapped scheduling after optimal retiming gives fastest implementation.

* Theory of transformations is still applicable to combinational logic in case of one-to-one mapping (think e.g. of converting the ripple-carry adder to the look-ahead adder by means of the look-ahead transformation).