SOFTWARE SYNTHESIS

Implementation of Digital Signal Processing

Sabih H. Gerez
University of Twente

PROJECT PROGRESS POLL

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Title</th>
<th>Max. points</th>
<th>Nominal Load</th>
<th>Start after lecture of</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAP</td>
<td>Mapping Data-Flow Graphs to RTL Designs</td>
<td>30</td>
<td>30 hours</td>
<td>February 19, 2021</td>
</tr>
<tr>
<td>TRA</td>
<td>Data-Flow-Graph Transformations</td>
<td>10</td>
<td>10 hours</td>
<td>March 5, 2021</td>
</tr>
<tr>
<td>GFS</td>
<td>The GFSK Receiver</td>
<td>60</td>
<td>60 hours</td>
<td>March 12, 2020</td>
</tr>
</tbody>
</table>

Please respond on your progress on the projects:

- A: Working on MAP
- B: MAP (almost) finished, not yet working on TRA
- C: Working on TRA
- D: TRA (almost) finished, not yet working on GFS
- E: Working on GFS

SOFTWARE SYNTHESIS

• Generation of executable code from data-flow graphs: single-processor schedules

• Used for:
  – Production software
  – Simulation software

• Based on following paper (all examples are taken from it):

THINK OF THE MULTIPLE CALLS TO THE run METHOD IN ARX C++ SIMULATIONS.
TOPICS

- Synchronous data flow (recap)
- Cyclo-static data flow
- Optimization criteria
- Vectorized schedules

SYNCHRONOUS DATA FLOW (SDF)

- Already discussed.
- Each firing of a node consumes a fixed number of tokens and produces a fixed number of tokens (these numbers are annotated along the edges).
- An edge can have delay (initial tokens).
- Consistency:
  - The repetitions vector (relative number of invocations for each node) should exist.
  - There should be no deadlock (situation where nodes are waiting for each other to produce tokens).

CONSISTENT SDF EXAMPLE

EXAMPLE OF SDF WITH DEADLOCK

- Easiest check for deadlock: simulation

$nD$ on an edge means, $n$ initial tokens.

4D on edge AB removes deadlock.
OPTIMIZATION CRITERIA

- Buffer memory
- Code memory
- Number of context switches

IMPLEMENTATION

- Inlined code
- Subroutines
- Hybrid

MINIMAL-BUFFER SCHEDULE

\[ S_1 = YZyZyZyZyZyZyZyZyZyZz \]

- Buffer size: \( \text{buf}(S_1) = 11 \)
- Code size: \( c_{\text{size}}(S_1) = \kappa(X) + 10\kappa(Y) + 10\kappa(Z) \)
- Context switches: \( c_{\text{sw}}(S_1) = 21 \)

MINIMAL-CODE-SIZE SCHEDULE (1)

\[ S_2 = (5YZ)X(5YZ) \]

- Buffer size: \( \text{buf}(S_2) = 11 \)
- Code size: \( c_{\text{size}}(S_2) \approx \kappa(X) + 2\kappa(Y) + 2\kappa(Z) \)
- Context switches: \( c_{\text{sw}}(S_2) = 21 \)

\[ S_3 = X(10Y)(10Z) \]

- Buffer size: \( \text{buf}(S_3) = 25 \)
- Code size: \( c_{\text{size}}(S_3) \approx \kappa(X) + \kappa(Y) + \kappa(Z) \)
- Context switches: \( c_{\text{sw}}(S_3) = 3 \)
MINIMAL-CODE-SIZE SCHEDULE (2)

\[ S_4 = X(10YZ) \]

- Buffer size: \( \text{buf}(S_4) = 16 \)
- Code size: \( \text{c\_size}(S_4) \approx \kappa(X) + \kappa(Y) + \kappa(Z) \)
- Context switches: \( \text{c\_sw}(S_4) = 21 \)