Implementing Modal Software in Data Flow for Heterogeneous Architectures

James Steed, Kerry Barnes,
William Lundgren
Gedae, Inc.
Core Gedae Data Flow

- Gedae’s Core Data Flow Relationships

<table>
<thead>
<tr>
<th></th>
<th>Number of Tokens Produced/Consumed</th>
<th>Restrictions on Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>static</td>
<td>Preplanned</td>
<td>Full Inputs/Empty Outputs</td>
</tr>
<tr>
<td>dynamic</td>
<td>Determined at Runtime</td>
<td>Full Inputs/Empty Outputs</td>
</tr>
<tr>
<td>nondet</td>
<td>Determined at Runtime</td>
<td>None</td>
</tr>
</tbody>
</table>

- Any application control can be implemented but
  - Complex modal software requires lots of logic
  - Done in an ad hoc manner that isn’t reusable
Stream Segmentation

- Infinite streams can be broken into finite length segments
- Segments are processed independently
- Primitives add segment begin and end markers to a data stream
- Each marker causes side effects downstream
Using Segmentation to Control Modes

- Segment markers cause old mode to end and new one to reset
- Exclusivity allows memory sharing between modes
Reset and EndOfSegment Methods

- Primitive code is grouped into methods
- When methods are executed:
  - Start: Beginning of execution
  - Reset: Beginning of each segment (start mode)
  - Apply: When queues are ready for execution (execute mode)
  - EndOfSegment: End of each segment (end mode)
  - Terminate: End of execution
Sharing Resources Between Modes

- Exclusivity: Only one output is actively producing a segment at any given time
- Subgraphs controlled by a family of exclusive outputs can share resources
  - Schedule memory
  - Queue memory
  - State variables
Moving static variable out of filterS subgraph causes it to be persistent between segment boundaries.

No transients due to clearing of static variables at segment boundaries.
Moving to Heterogeneity

- Gedae relies on
  - C cross compilers
  - Optimized vector libraries
to run on DSPs.

- How do we support firmware targets like FPGAS?
Single Sample Language: Gedae-RTL

- Single sample extension to Gedae graph language
- Based on the theory of register transfer languages
- Registers store information, delayed by a clock rate

\[
out(i) = K \times (in(i) - out(i-1)) + \text{out}(i-1)
\]
Gedae-RTL’s Seven Functions

• **Register** \( R(\text{in}, \text{out}, c) \)
  - Copy \text{in} to \text{out} delayed by clock rate \( c \).

• **Assignment** \( A(E, \text{out}) \)
  - Evaluate the expression \( E \) and assign its value to \text{out}.

• **Decimate** \( D(\text{in}, c) \)
  - Tie clock rate \( c \) to signal \text{in}.

• **Clock** \( C(\text{in}, c) \)
  - Get clock rate \( c \) tied to \text{in}.

• **Memory** \( M(\text{in}, n, s) \)
  - Allocate buffer \text{in} with \( n \) elements of size \( s \).

• **Read** \( MR(a, \text{out}) \)
  - Read the element at address \( a \) and put the value in \text{out}.

• **Write** \( MW(\text{in}, a) \)
  - Write the value \text{in} to address \( a \).
Language Independence

- Language Support Package (LSP) allows exportation of target-specific code
- Export Ansi-C to simulate functionality
- Export VHDL for FPGAs
- Export C enhanced for the AltiVec architecture
Example: 5-Point FIR Filter

Pipeline of N R(\() \text{ registers}

Set of N A(a*b, out)

Network of N-1 A(a+b, out)

\[
\text{out}(i) = C0*\text{in}(i) + C1*\text{in}(i-1) + C2*\text{in}(i-2) \ldots
\]
Gedae Implements the Heterogeneous System

- Pin connections to camera and VGA provide real time I/O
- Host updates threshold in non-real time

Gedae-RTL graphs on the FPGA perform Sobel in real time

Gedae graphs on the DSP calculate new thresholds