SHIM: A Deterministic Model for Heterogeneous Embedded Systems

Stephen A. Edwards and Olivier Tardieu
Department of Computer Science, Columbia University
www.cs.columbia.edu/~sedwards
{sedwards,tardieu}@cs.columbia.edu
Definition

**shim**  
ˈshim\ n

1: a thin often tapered piece of material (as wood, metal, or stone) used to fill in space between things (as for support, leveling, or adjustment of fit).

2: *Software/Hardware Integration Medium*, a model for describing hardware/software systems
Conclusions

SHIM is an effective model of computation for embedded hardware/software systems.

Formal semantics guarantee determinism & boundedness.

Easy to synthesize into hardware and software.

Applicable to large, important class of systems, but not all.

Embedded systems should be designed on the SHIM model of computation.
Robby Roto (Bally/Midway, 1981)
HW/SW Interaction

Software → Blitter → Memory → Video

- Blit
- Pixels
- Line
- Blit
- Pixels
- Line
- Blit
- Pixels
- Line
- Blit
- Interrupt
- Line
SHIM Wishlist

- *Mixes synchronous and asynchronous styles*
  Need multi-rate for hardware/software systems

- *Delay-insensitive (Deterministic)*
  Want simulated behavior to reflect reality
  Verify functionality and performance separately

- *Only requires bounded resources*
  Hardware resources fundamentally bounded

- *Formal semantics*
  Do not want arguments about what something means
The SHIM Model

- Sequential processes
- Unbuffered point-to-point communication channels exchange data tokens
- Fixed topology
- Asynchronous
- Synchronous communication events
- Delay-insensitive: sequence of data through any channel independent of scheduling policy (the Kahn principle)

“Kahn networks with rendezvous communication”
## SHIM vs. Other Models

<table>
<thead>
<tr>
<th>Feature</th>
<th>SHIM</th>
<th>CSP</th>
<th>Kahn</th>
<th>SDF</th>
<th>Haste</th>
<th>Sync</th>
<th>Petri</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deterministic</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Blocking</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Communication</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Bounded Buffers</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Multi-Rate</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Data-Dependent Rates</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Easy-To-Schedule</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Static Scheduling</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
# Modeling in SHIM

<table>
<thead>
<tr>
<th>To model</th>
<th>introduce</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffers</td>
<td>Buffer processes</td>
</tr>
<tr>
<td>Interrupts</td>
<td>Polling and periodic communication</td>
</tr>
<tr>
<td>Synchrony</td>
<td>Clock signals</td>
</tr>
<tr>
<td>Synchronous dataflow</td>
<td>Buffers</td>
</tr>
<tr>
<td>Sensors</td>
<td>Source processes</td>
</tr>
<tr>
<td>Arbiters</td>
<td>A deterministic algorithm</td>
</tr>
</tbody>
</table>
Modeling Time in SHIM

SHIM is timing-independent

Philosophy: separate functional requirements from performance requirements

Like synchronous digital logic: establish correct function independent of timing, then check and correct performance errors

Vision: clock processes impose execution rates, checked through static timing analysis
The Syntax of Tiny-SHIM

<table>
<thead>
<tr>
<th>Expressions</th>
<th>Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e ::= L$</td>
<td>$s ::= V = e$ (assignment)</td>
</tr>
<tr>
<td>$V$</td>
<td>if ($e$) $s$ else $s$ (conditional)</td>
</tr>
<tr>
<td>$op e$</td>
<td>while ($e$) $s$ (loop)</td>
</tr>
<tr>
<td>$e op e$</td>
<td>$s ; s$ (sequencing)</td>
</tr>
<tr>
<td>( $e$ )</td>
<td>{ $s$ } (grouping)</td>
</tr>
<tr>
<td>( $e$ )</td>
<td>read($C, V$) (blocking read)</td>
</tr>
<tr>
<td></td>
<td>write($C, e$) (blocking write)</td>
</tr>
</tbody>
</table>
Local variables: \(d, e\)

\[
d = 0; \\
while (1) { \\
    e = d; \\
    while (e > 0) { \\
        write(c, 1); \\
        write(c, e); \\
        e = e - 1; \\
    } \\
    write(c, 0); \\
    d = d + 1; \\
}
\]

Local variables: \(a, b, r, v\)

\[
a = 0; \\
b = 0; \\
while (1) { \\
    r = 1; \\
    while (r) { \\
        read(c, r); \\
        if (r != 0) { \\
            read(c, v); \\
            a = a + v; \\
        } \\
    } \\
    b = b + 1; \\
}
\]
while the player is alive do
  Wait for start-of-frame
  ...game logic...
  Write "false" to end-of-frame
  Write to the blitter
while 1 do
  while not end-of-frame do
    Read blit command
    Write pixels to memory
  Write frame
while 1 do
  Write start-of-frame
  for each line do
    Emit line timing signals
    for each pixel do
      Wait for pixel clock
      Read pixel from memory
      Send pixel to display
Read next frame
The SOS Semantics of Tiny-SHIM

- $\sigma$: Process memory state
- $\langle \sigma, p \rangle$: Process $p$ in state $\sigma$
- $p$: Process code
- $\mathcal{E}(\sigma, e)$: Value of $e$ in $\sigma$
- $\langle \sigma, v = e \rangle \rightarrow \langle \sigma[v \leftarrow n] \rangle$ (assign)
- $\langle \sigma, \text{if} (e) \ p \ \text{else} \ q \rangle \rightarrow \langle \sigma, p \rangle$ (if-true)
- $\langle \sigma, \text{if} (e) \ p \ \text{else} \ q \rangle \rightarrow \langle \sigma, q \rangle$ (if-false)
- $\langle \sigma, \text{while} (e) \ p \rangle \rightarrow \langle \sigma, p ; \text{while} (e) \ p \rangle$ (while-true)
- $\langle \sigma, \text{while} (e) \ p \rangle \rightarrow \langle \sigma \rangle$ (while-false)
- $\langle \sigma, \text{read} (c, v) \rangle \xrightarrow{\text{c get } n} \langle \sigma[v \leftarrow n] \rangle$ (read)
- $\langle \sigma, \text{write} (c, e) \rangle \xrightarrow{\text{c put } n} \langle \sigma \rangle$ (write)
- $\mathcal{E}(\sigma, e) = n$
- $\langle \sigma, p \rangle \rightarrow s$
- $\{\langle \sigma, p \rangle\} \cup S \Rightarrow \{s\} \cup S$ (step)
- $\langle \sigma, p \rangle \xrightarrow{\text{c put } n} s$
- $\langle \sigma', p' \rangle \xrightarrow{\text{c get } n} s'$ (sync)
Syntax-Directed HW Translation

\[
\text{if}(e) \ s_1 \ \text{else} \ s_2
\]

\[
\text{while}(e) \ s
\]

\[
\text{write}(c, e)
\]

\[
\text{read}(c, v)
\]
Hardware Translation Example

d = 0;
while (1) {
    e = d;
    while (e > 0) {
        write(c, 1);
        write(c, e);
        e = e - 1;
    }
    write(c, 0);
    d = d + 1;
}

a = 0;
b = 0;
while (1) {
    r = 1;
    while (r) {
        read(c, r);
        if (r != 0) {
            read(c, v);
            a = a + v;
        }
    }
    b = b + 1;
}
Ongoing Work

- Translation into software
- Relaxation of block-on-single-channel rule
- Complete hardware/software design language
- Translation optimization for hardware and software