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
Robby Roto (Bally/Midway, 1981)
HW/SW Interaction

- Software
- Blitter
- Memory
- Video

- Interrupt
- Blit
- Pixels
- Line

- Blit
- Pixels
- Line

- Blit
- Pixels
- Line

- Interrupt
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
Deterministic, Concurrent MoCs

Not too many:

The Synchronous Model
Bad for multi-rate and asynchronous behavior

The Lambda Calculus
Unbounded in general, not obvious in hardware

Kahn Networks
Unbounded in general, difficult to schedule

Idea: Restrict Kahn to be bounded.
The SHIM Model

Kahn networks with rendezvous communication

Sequential processes

Unbuffered point-to-point communication channels exchange data tokens

Fixed communication topology

Fundamentally asynchronous

Each communication event is synchronous (like a clock)

Delay-insensitive: sequence of data through any channel is independent of scheduling policy (the Kahn principle)
Tiny-SHIM Processes

Local variables: d, e

```c
int d = 0;
while (1) {
    int 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

```c
int a = 0;
int b = 0;
while (1) {
    int r = 1;
    while (r) {
        read(c, r);
        if (r != 0) {
            read(c, v);
            a = a + v;
        }
    }
    b = b + 1;
}
```
Behavior of the Processes

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;
}
while the player is alive do
  Wait for start-of-frame
  ...game logic...
  Write “false” to end-of-frame
  Write to the blitter
  ...game logic...
  Write “true” to end-of-frame

while 1 do
  while not end-of-frame do
    Read blit command
    Write pixels to memory
    Write frame
  end-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
  frame
  Write start-of-frame
  Read next frame
The Syntax of Tiny-SHIM

\[ e ::= L \quad \text{(literal)} \quad s ::= V = e \quad \text{(assignment)} \]

\[ V \quad \text{(variable)} \quad \text{if} ( e ) s \text{ else } s \quad \text{(conditional)} \]

\[ op \ e \quad \text{(unary op)} \quad \text{while} ( e ) s \quad \text{(loop)} \]

\[ e \ op \ e \quad \text{(binary op)} \quad s ; s \quad \text{(sequencing)} \]

\[ ( e ) \quad \text{(paren)} \quad \text{read} ( C, V ) \quad \text{(blocking read)} \]

\[ \text{write} ( C, e ) \quad \text{(blocking write)} \]

\[ \{ s \} \quad \text{(grouping)} \]
The SOS Semantics of Tiny-SHIM

\[ \sigma \quad \text{Process memory state} \quad p \quad \text{Process code} \]

\[ \langle \sigma, p \rangle \quad \text{Process } p \text{ in state } \sigma \quad \langle \sigma \rangle \quad \text{Terminated in state } \sigma \]

\[ a \rightarrow \quad \text{Single-process rule} \quad \Rightarrow \quad \text{System rule} \]

\[ E(\sigma, e) \quad \text{Value of } e \text{ in } \sigma \]

\[ \frac{E(\sigma, e) = n}{\langle \sigma, v = e \rangle \rightarrow \langle \sigma[v \leftarrow n] \rangle} \quad \text{(assign)} \]

\[ \frac{E(\sigma, e) \neq 0}{\langle \sigma, \text{if } (e) \ p \ \text{else } q \rangle \rightarrow \langle \sigma, p \rangle} \quad \text{(if-true)} \]

\[ \frac{E(\sigma, e) = 0}{\langle \sigma, \text{if } (e) \ p \ \text{else } q \rangle \rightarrow \langle \sigma, q \rangle} \quad \text{(if-false)} \]
Semantics of Looping & Sequencing

\[ \varepsilon(\sigma, e) \neq 0 \quad \frac{\langle \sigma, \textbf{while } (e) \ p \rangle \rightarrow \langle \sigma, p ; \textbf{while } (e) \ p \rangle}{(\text{while-true})} \]

\[ \varepsilon(\sigma, e) = 0 \quad \frac{\langle \sigma, \textbf{while } (e) \ p \rangle \rightarrow \langle \sigma \rangle}{(\text{while-false})} \]

\[ \frac{\langle \sigma, p \rangle \xrightarrow{a} \langle \sigma', p' \rangle}{\langle \sigma, p ; q \rangle \xrightarrow{a} \langle \sigma', p' ; q \rangle} \quad (\text{seq}) \]

\[ \frac{\langle \sigma, p \rangle \xrightarrow{a} \langle \sigma' \rangle}{\langle \sigma, p ; q \rangle \xrightarrow{a} \langle \sigma', q \rangle} \quad (\text{seq-term}) \]
Communication and Concurrency

\[
\langle \sigma, \text{read}(c, v) \rangle \xrightarrow{c \text{ get } n} \langle \sigma[v \leftarrow n] \rangle \quad \text{(read)}
\]

\[
\mathcal{E}(\sigma, e) = n \\
\langle \sigma, \text{write}(c, e) \rangle \xrightarrow{c \text{ put } n} \langle \sigma \rangle \quad \text{(write)}
\]

\[
\langle \sigma, p \rangle \rightarrow s \\
\{\langle \sigma, p \rangle \} \cup S \Rightarrow \{s\} \cup S \quad \text{(step)}
\]

\[
\langle \sigma, p \rangle \xrightarrow{c \text{ put } n} s \\
\langle \sigma', p' \rangle \xrightarrow{c \text{ get } n} s' \\
\{\langle \sigma, p \rangle, \langle \sigma', p' \rangle \} \cup S \Rightarrow \{s, s'\} \cup S \quad \text{(sync)}
\]
Translating Tiny-SHIM to Hardware

Assignment

Decision

Merge

Cycle Boundary

CFG Node

Control Fragment

Datapath Fragment

\[ v = e \]
Translation Patterns

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

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

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

\[ \text{read}(c, v) \]
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;
}
Summary

- SHIM: A delay-insensitive (deterministic) model of computation that supports synchrony and asynchrony
- Tiny-SHIM: A little language that embodies the model
- Formal operational semantics of Tiny-SHIM
- A procedure for translating Tiny-SHIM into hardware
Ongoing Work

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