Translating SMALL Programs to FPGA Configurations

Ying Shen & Theodore S. Norvell
Memorial University of Newfoundland
NECEC 1999
SMALL

SMALL is ...

- An imperative programming language
- *Synchronous* and *parallel*
- A low level language — compared to C
- A high level language — compared to VHDL
- Intended for hardware design

The SMALL implementation

- From program to hardware at a keypress.
- Short design cycles
- Reasonable efficiency
- Thesis:
  * hardware is becoming very cheap
  * hardware is programmed more than manufactured
  * programmed hardware is getting faster faster

Design costs will dominate.
The elements of SMALL

Types and Expressions

- Booleans — Logic expressions
- N-Dimensional Arrays — Arithmetic & APLish expressions

Entities

- Registers — Record values
- Signals — Transfer values to parallel statements.

Statements

<table>
<thead>
<tr>
<th>Assign Register:</th>
<th>$r \leftarrow \text{Exp}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assert Signal:</td>
<td>$s \mid \text{Exp}$</td>
</tr>
<tr>
<td>Wait for clock:</td>
<td>$\text{tick}$</td>
</tr>
<tr>
<td>Parallel composition:</td>
<td>par $S \parallel T \parallel U \text{ rap}$</td>
</tr>
<tr>
<td>Sequential composition:</td>
<td>$S T U$</td>
</tr>
<tr>
<td>Choice:</td>
<td>if $\text{Exp}$ then $S$ else $T$ fi</td>
</tr>
<tr>
<td>Loops:</td>
<td>while $\text{Exp}$ do $S$ od</td>
</tr>
<tr>
<td>Loops:</td>
<td>repeat $S$ until $\text{Exp}$</td>
</tr>
</tbody>
</table>
Timing semantics

Time passes only

• at *tick* statements
• after loop iterations
• for parallel process waiting for another to terminate.

\[
s!_{10} \text{ as 4 bits} \\

t! s - u
\]

is identical to

\[
t! s - u \\
s!_{10} \text{ as 4 bits}
\]

But

\[
s!_{10} \text{ as 4 bits} \\
tick \\
t! s - u
\]

is NOT identical to

\[
t! s - u \\
tick \\
s!_{10} \text{ as 4 bits}
\]
Implementations

- Simulator — for debugging of designs
- Compiler — for hardware implementation

Implementation block diagram
Parallel Algorithmic State Machine Charts (PASM Charts)

A state-machine representation for control flow.

Example SMALL program

```
global sig in : bool
global sig out : bool
reg r : bool
par
  while true
do r ← in
  od
||
  while true
do if r
    then out ! in
    fi
  od
rap
```
The state is represented by a set of active state-nodes. Nodes reachable from an active state-node are executed. Then all state nodes reachable from an active state-node become active.
Translation of signals and asserts

\[ \text{sig } s : \text{ bool} \]

\[ \text{go0} \rightarrow \text{done0} \]
\[ \text{go1} \rightarrow \text{done1} \]

\[ \begin{align*}
0 & \quad \text{s!E} \\
1 & \quad \text{s!F}
\end{align*} \]
Sets and assignments

\( r \leftarrow E \)

\( r \leftarrow F \)

\text{reg } s : \text{bool}

\text{change } r

\text{D}

\text{go2} \rightarrow \text{done2}

\text{go3} \rightarrow \text{done3}
Dealing with Control State
Optimization

- Dead device and wire elimination
  - Deadwood is removed
- Constant folding and propagation
  - Ground and power inputs eliminated
- Common subcircuit elimination
  - Two devices with the same inputs
- PASM chart level — not yet done
  - Sharing of nodes and expressions
Future Directions

• Language improvements
  ∗ Asynchronous communication.
  ∗ Module system and separate compilation
  ∗ Higher level types
  ∗ User defined types

• Compiler improvements
  ∗ Speed and Space
  ∗ More optimization — resource sharing.

• Industry involvement