SEQLAB AND HCL

CS429H - Spring 2011

Christian Miller

SEQLAB

- We give you a bigger, better Y86 simulator
- You modify it to include two new instructions
  - iaddl and leave
- You do this by modifying the HCL description

HCL

- A toy hardware description language
  - Fake, actually compiles into a C program
- Looks a lot like C
- Does not execute sequentially, but simultaneously
  - Think logic gates, not assembly
  - Do not create loops!

Data

- Only two types: bool (a single bit) and int (32 bits)
- boolsig and intsig tie into the simulator
    - That means don't edit them
- Equals doesn't assign, it renames
    - Basically attaches names to wires

Expression syntax

Syntax	Meaning
`0`	Logic value 0
`1`	Logic value 1
name	Named Boolean signal
int-expr `in` {*int-expr*$_1$`,` *int-expr*$_2$`,` ... `,` *int-expr*$_k$}	Set membership test
int-expr$_1$ `==` *int-expr*$_2$	Equality test
int-expr$_1$ `!=` *int-expr*$_2$	Not equal test
int-expr$_1$ `<` *int-expr*$_2$	Less than test
int-expr$_1$ `<=` *int-expr*$_2$	Less than or equal test
int-expr$_1$ `>` *int-expr*$_2$	Greater than test
int-expr$_1$ `>=` *int-expr*$_2$	Greater than or equal test
`!` *bool-expr*	NOT
bool-expr$_1$ `&&` *bool-expr*$_2$	AND
bool-expr$_1$ `||` *bool-expr*$_2$	OR

Expression semantics

- Can be nested using parentheses
- Set membership returns true if something is in a given set
- Expressions basically compile into logic tables
- No partial evaluation, everything is evaluated

Case syntax

$$
\begin{array}{[lcl]}
\textit{bool-expr}_1 & : & \textit{int-expr}_1 \\
\textit{bool-expr}_2 & : & \textit{int-expr}_2 \\
 & \vdots & \\
\textit{bool-expr}_k & : & \textit{int-expr}_k
\end{array}
$$

Case semantics

- Think switch statement
- Effectively compiles into a mux (output selector)
- Internally wrangled to evaluate in order
    - You can throw in “1 :” as a default at the end

Go forth

- You have an embarrasingly long time to do this.
- It can be done in 15 lines... easily.