SANDBOX

Megan Fillion, Gabriel Guzman, and Dimitri Leggas

mlf2179, grg2117, and ddl2133

Overview

- Motivation
  - Improve our understanding of digital systems
  - Simple HDL to facilitate our/others' learning
  - A challenging PLT project

- Goals
  - Simple and easy to code HDL for programming students learning about digital systems.
  - Python like syntax; Scope determined by indentation
  - Succinct with shorthand syntax (more later)
  - Functional flavor to the language

Tutorial

- Functions represent circuit blocks
  - map a list of input busses to a list of output busses
- Busses represent *k*-bit integers
- Start off with the function sandbox
  - Main executive function
  - Inputs and outputs of sandbox function are the io of the circuit
  - Builds the circuit through calls to other blocks
- The clock is internal and implicit

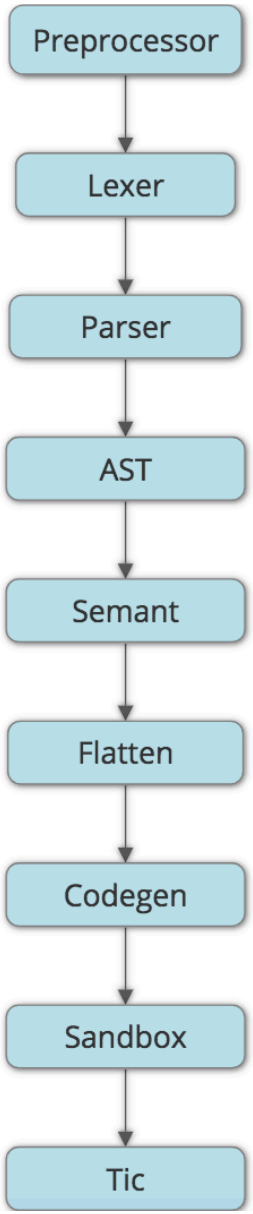
Simple Sample Code

```
/ our hello world /
(bit a, bit b, bit cin) sandbox (bit s, bit c):
        a ^ b ^ cin -> s
        (a & b) ^ (cin & (a ^ b)) -> c

**********************************************************************

/ simple counter /
( ) sandbox ( bit s ):
        s + 1 -: s
```

Compiler Structure


Preprocessor
Lexer
Parser
AST
Semant
Flatten
Codegen
Sandbox
Tic

Flatten

- Collapses sandbox program into list of outputs in terms of inputs
- Recursive walk over function calls
  - Maps actual inputs to formal inputs and formal outputs to actual outputs

```
/ flattening a function call /
(bit x, bit y) halfadder (bit w, bit z):
        x ^ y -> w
        x & y -> z
(bit a, bit b) sandbox (bit s, bit c):
        [a, b] halfadder [s, c]
```

a b ^ s -> a b & c ->

Flatten Fell Flat

- Also needed to break busses into operations on single bits and support shorthand function calls; maybe in the next 24hrs!!!!!!

```
/ what we wanted it to look like /
(bit a, bit b, bit cin) fulladder (bit s, bit c):
        a ^ b ^ cin -> s
        (a & b) ^ (cin & (a^b)) -> c

(bit a.4, bit b.4, bit cin) sandbox (bit sum.4, bit cout.4):
        [a, b, cin::cout(0:3) ] fulladder [sum, cout]
```

Codegen

- Translates post-order traversal given by flatten into a single LLVM function
  - Pushes literals and variables from the flattened list onto a stack and pops them as operations and assignments are encountered in order to build LLVM statements
- Sandbox allows multiple returns
  - The function created in LLVM takes a pointer to the inputs and outputs
  - indexes the memory in both arrays, loads the inputs at the beginning, stores the outputs at the end
- Sequential Logic
  - Keeps track of states by allocating two static LLVM variable for each sandbox variable
  - If sandbox is called with state 0, load from 0 and store in 1

Tic

- Simple function written in C to call the function generated in LLVM inside of a loop, printing outputs at each step
- Defines: extern void sandbox(int* ins, int* outs, int state)
- Build an executable for a sandbox file by compiling it to bytecode and then compiling: gcc –o name tic.c name.s

Lessons Learned

- Teamwork is hard and different parts of projects depended on others
- Everything took longer than we thought
- Former project code on Edward’s website was immensely helpful
- Written test cases helped to find bugs and improve our understanding of semantics
- Improved our understanding of version control systems
- Pick a smaller project next time!