Constructive Computer Architecture

Tutorial 1

Andy Wright
6.S195 TA

Administrative Stuff

- Piazza
  - Was everyone added to the class?
- Lab 1 (Due Friday)
  - Has everyone tried logging onto vlsifarm?
  - What about checking out the lab git repositories?
  - Anyone having technical problems?

Bluespec Introduction

- Combinational Circuits
  - Written as Functions
    - Problem: all functions inlined, no code reuse
  - Written as Modules
    - Allows for code reuse, but is more complicated
- Sequential Circuits
  - Requires Reg#(…)'s and Rules
    - Currently the examples from lecture only use 1 rule.
    - Adding multiple rules compilcates things and will be covered in a later lecture.

Bluespec Types

- Bool
- Bit#(n)
- Enum
- Struct

```
typedef enum {
    ADD,
    SUB,
    MUL,
    DIV
} Operation deriving(Bits, Eq);
```

```
typedef struct {
    Operation op;
    Bit#(32) val;
} Command deriving(Bits, Eq);
```

Bluespec Type - Tuples

- Type: Tuple2#(type a, type b)
  - Also Tuple3#(a,b,c) up to Tuple8#(…)
- Values: tuple2(val_a, val_b)
- Components: tpl_1(t), tpl_2(t), …

```
Tuple2#(Bool, Bit#(16)) t = tuple2( true, 4 );
Bool a_val = tpl_1(t);
Bit#(16) b_val = tpl_2(t);
```


Bluespec Type - Vectors

- Type: Vector#(numeric type size, type data_type)
- Values: newVector(), replicate(val)
- Elements accessed by []
- Advanced functions like zip, map, and fold
  - See the Bluespec Reference Manual for more info

```
Vector#(32, Bit#(32)) rfile_values = replicate(0);
rfile_values[7] = 5;
```

Bluespec Type - Reg

- Type: Reg#(type data_type)
- Instantiated differently from normal variables
  - Uses <- notation
- Written to differently from normal variables
  - Uses <= notation

```
Reg#(Bit#(32)) a_reg <- mkReg(0) // value set to 0
Reg#(Bit#(32)) b_reg <- mkRegU() // uninitialized

// write to b_reg (needs to be done inside rule)
b_reg <= 7;
```


Reg and Vector

- Register of Vectors
  - Reg#(Vector#(32, Bit#(32))) rfile;
  - rfile <- mkReg(replicate(0));
- Vector of Registers
  - Vector#(32, Reg#(Bit#(32))) rfile;
  - rfile <- replicateM(mkReg(0));
- Each has its own advantages and disadvantages

Numeric Types

```
typedef 5 N; // N is the numeric type 5
Bit#(N); // Same as Bit#(5)
valueOf(N); // The Integer 5

Bit#(n); // type variable
```


Modules

- Modules are building blocks for larger systems
  - Modules contain other modules and ruls
  - Moduels are accessed through their interface
- `module mkAdder( Adder#(32) );`
  - Adder#(32) is the interface

Interfaces

- Interfaces contain methods for other modules to interact with the given module
  - Interfaces can also contain other interfaces

```
interface MyInterface#(numeric type n);
    method ActionValue#(Bit#(b)) f();
    interface SubInterface s;
endinterface
```


Interface Methods

- Method
  - Returns value, doesn't change state
  - `method Bit#(32) peek_at_front();`
- Action
  - Changes state, doesn't return value
  - `method Action enqueue();`
- ActionValue
  - Changes state, returns value
  - `method ActionValue#(Bit#(32)) dequeue_front()`

Strong Typing

- The Bluespec Compiler throws errors if it can't figure out a type
- Which of the following lines work?

```
Bit#(32) a = 7;  Bit#(8) small_b = 3;

let b = zeroExtend( small_b );

Bit#(32) other_b = zeroExtend( small_b );

let a_plus_b = a + zeroExtend( small_b );

Bit#(8) small_b_plus_fifty_truncated
    = truncate( 50 + zeroExtend( small_b ) );
```

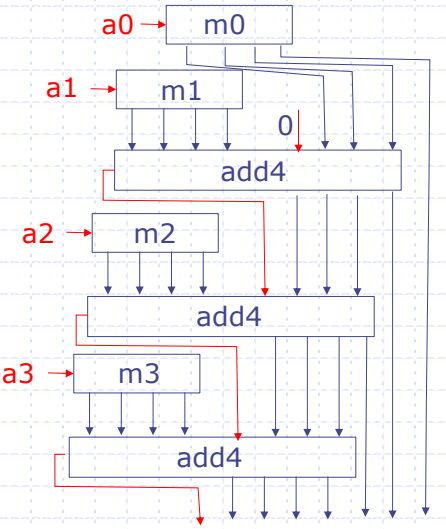

WideMux Example

Addition Circuits and Critical Paths

Multiplication by repeated addition

```
b Multiplicand  1101   (13)
a Muliplier   * 1011   (11)
tp              0000
m0          +   1101
tp            01101
m1          +   1101
tp           100111
m2          +  0000
tp            0100111
m3          + 1101
tp           10001111  (143)
```

```
mi = (a[i]==0)? 0 : b;
```



Design issues with combinational multiply

- Lot of hardware
  - 32-bit multiply uses 31 add32 circuits
- Long chains of gates
  - 32-bit ripple carry adder has a 31-long chain of gates
  - 32-bit multiply has 31 ripple carry adders in sequence!

The speed of a combinational circuit is determined by its longest input-to-output path

What is this path?