

# Using Functional Programming Languages in IBIS-AMI Modeling

David Banas

IBIS Open Forum Summit at DesignCon,  
Santa Clara

February 2, 2012

Version: 1.0



# Presentation Outline

- What are functional languages?
- Why functional languages?
- Where/how are/can they be used?
- How do they compare to C?
- Q&A

# Presentation Outline

- What are functional languages?
- Why functional languages?
- Where/how are/can they be used?
- How do they compare to C?
- Q&A

# What are *functional* languages?

- They are a means of expressing a desired computation, at a higher level of abstraction than is provided by *imperative* languages.
- Consider the generation of the Fibonacci sequence:

```
// Fibonacci numbers, imperative style (C++, 10 lines)
int fibonacci(int iterations) {
    int first = 0, second = 1; // seed values
    for (int i = 0; i < iterations; ++i) {
        int sum = first + second;
        first = second;
        second = sum;
    }
    return first;
}
std::cout << fibonacci(10) << "\n";
```

```
-- Fibonacci numbers, functional style (Haskell, 3 lines)
-- describe an infinite list based on the recurrence relation for Fibonacci numbers
fibRecurrence first second = first : fibRecurrence second (first + second)

-- describe fibonacci list as fibRecurrence with initial values 0 and 1
fibonacci = fibRecurrence 0 1

-- describe action to print the 10th element of the fibonacci list
main = print (fibonacci !! 10)
```

# What are *functional* languages? (cont'd.)

- Consider the following properties of the preceding functional code:
  - It is *succinct*. (3 vs. 10 lines)
  - It expresses the *functional intent* of the computation, as opposed to the (one of many possible) *series of steps* needed to implement the desired computation.
  - It is allowed to define an *infinite* sequence.
  - It does not require the intermediate `sum` variable of the C++ code, nor does it need to explicitly shuttle values between `first`, `second`, and `sum` at each loop iteration. That is, it is free of *intermediate state maintenance*. And it requires no mention of any variables, other than the original inputs to the function.
- Some newer imperative languages have functional constructs. (i.e. – Perl, Python)

# Presentation Outline

- What are functional languages?
- Why functional languages?
- Where/how are/can they be used?
- How do they compare to C?
- Q&A

# Why functional languages?

## ■ Succinctness

- Current AMI parameter parsing code lengths:
  - C - 800+ lines
  - Haskell - 80 lines

## ■ Believability

- Strong/static, as opposed to weak/dynamic, typing makes code that compiles easy to believe.

## ■ Higher abstraction level

- Tend to describe what you want done, rather than how the CPU should do it.
- Makes intent of code easier to understand.
- Leaves compiler free to optimize implementation for different targets.

# Presentation Outline

- What are functional languages?
- Why functional languages?
- **Where/how are/can they be used?**
- How do they compare to C?
- Q&A

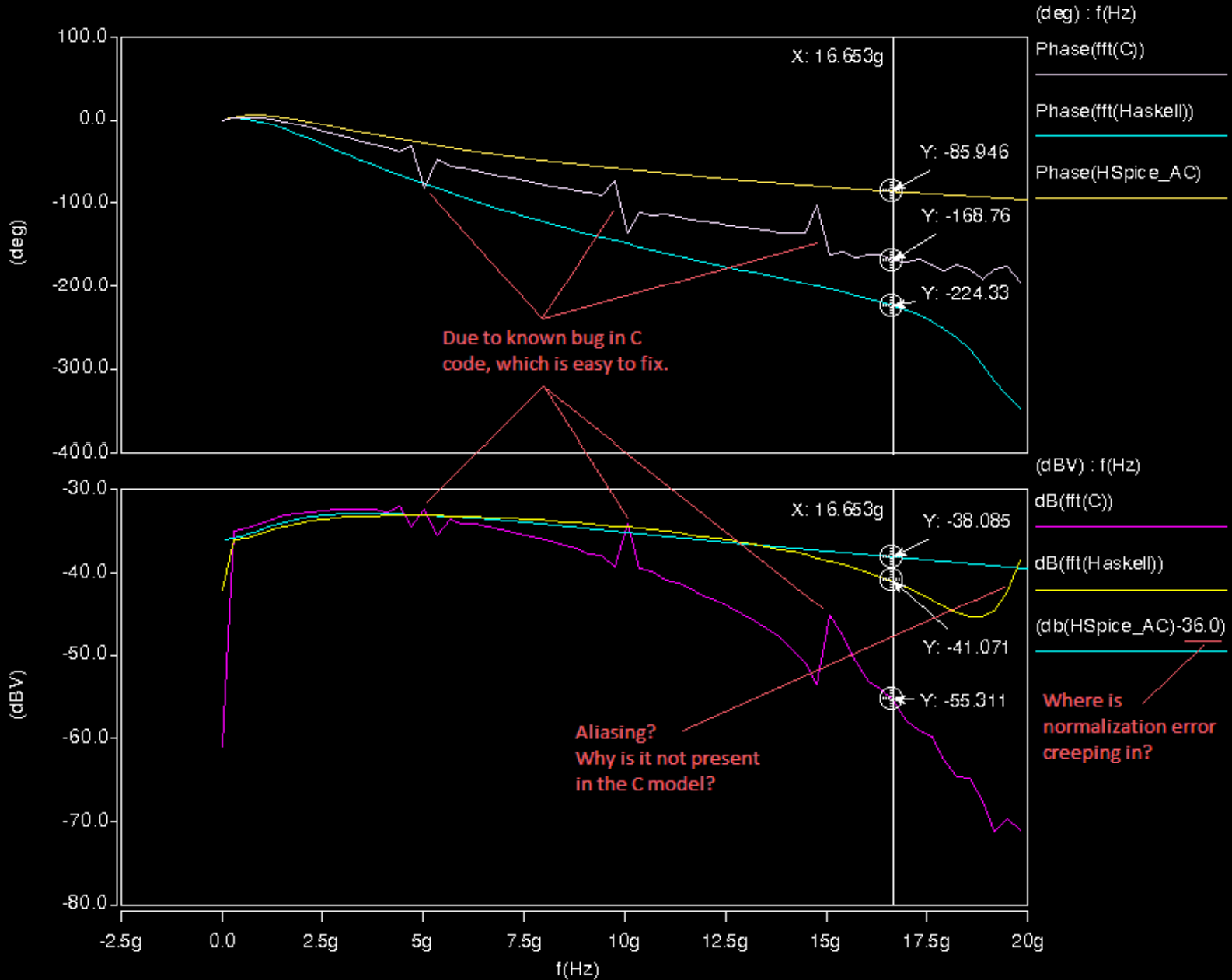
# Where/how are/can they be used?

- NOT (yet) as source of released model.
  - Still too obscure. (support liability to enterprise)
  - Performance tuning is still a black art.
- As executable specification.
  - Higher abstraction level yields something in between natural language and C, in terms of readability.
  - Natural languages can't be used as a verification metric by a scripted/automated process; functional programming languages can.
- As architectural exploration tool.
  - Functional programs tend to be quick to create/modify, due to their succinct and expressive nature.
  - Believability allows high level “what if”s to be explored with confidence.

# Presentation Outline

- What are functional languages?
- Why functional languages?
- Where/how are/can they be used?
- How do they compare to C?
- Q&A

Comparison of C and Haskell IBIS-AMI Rx CTLE Filter Responses to HSpice AC Sweep Results



## Notes on previous graph

- The agreement between the Haskell model and Hspice AC sweep, with regard to spectral magnitude, is excellent, out past 16 GHz.
- The spurs in the C model transfer function at 5, 10, and 15 GHz are due to a well understood bug in the C code, which is easy to fix.
- The flipped up tail near the end of the Haskell model spectral magnitude curve is reminiscent of aliasing, but I'm puzzled as to why the same effect is not present in the C model.
- Finally, it is necessary to offset the HSpice results by -36 dB, in order to match the three models. I don't understand where this normalization error is creeping into the process.

# Haskell vs. C Performance Comparison

- (Stay tuned...)

# Presentation Outline

- What are functional languages?
- Why functional languages?
- Where/how are/can they be used?
- How do they compare to C?
- Q&A

# Thank You

© 2012 Altera Corporation

ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at [www.altera.com/legal](http://www.altera.com/legal).

The Altera logo is rendered in a white, stylized, outlined font. The letters are bold and blocky, with a registered trademark symbol (®) positioned to the upper right of the final letter 'A'. The logo is set against a dark blue background that features a decorative graphic of wavy, overlapping bands in various shades of blue and orange, creating a sense of motion and depth.