Institute *for* System Level Integration

THE ALBA CAMPUS

NALLATECH

An Elementary Transcendental Function Core Library for Reconfigurable Computing

Robin Bruce, Dr Malachy Devlin, Prof Stephen Marshall

Introduction

- Project: Implement Floating-Point Math Functions on FPGAs
- Motivation: Transcendental Functions are well-suited to FPGAs

1. What is the Nallatech Math Library
2. What should I know about developing a core library?
3. What should I know about developing a math library?

Present How to Implement a Math Library

Nallatech Math Library

- Targeted at DIME-C
- Mostly single-precision subset of math.h
  - Elementary Functions
  - Random Number Generators
  - Utility Functions
- Precision
  - Double Precision
  - Aiming For Faithfully Rounded, Didn't Always get there
- Collection of ngc files, xml descriptor file and C header file

math.h

- Implemented so far:
  - float expf(float x);
  - float logf(float x);
  - float sinf(float x);
  - float cosf(float x);
  - float tanf(float x);
  - float powf(float x, float y);
  - float frexpf(float value, int exp);
  - float ldexpf(float x, int n);
  - int rand_ms();
  - int rand_ms_i(int seed, bool mode);
  - int rand();
  - double frexp(double value, int exp);
  - double ldexp(double x, int n);

Example Core Architecture – Exponential Function

- Limited Input Range: [-88,88]
- Floating Point Converted to Fixed Point
- Split into integer and fraction
  - Integer indexes lookup (512 SP words in a BRAM)
  - Fraction input to polynomial approximation

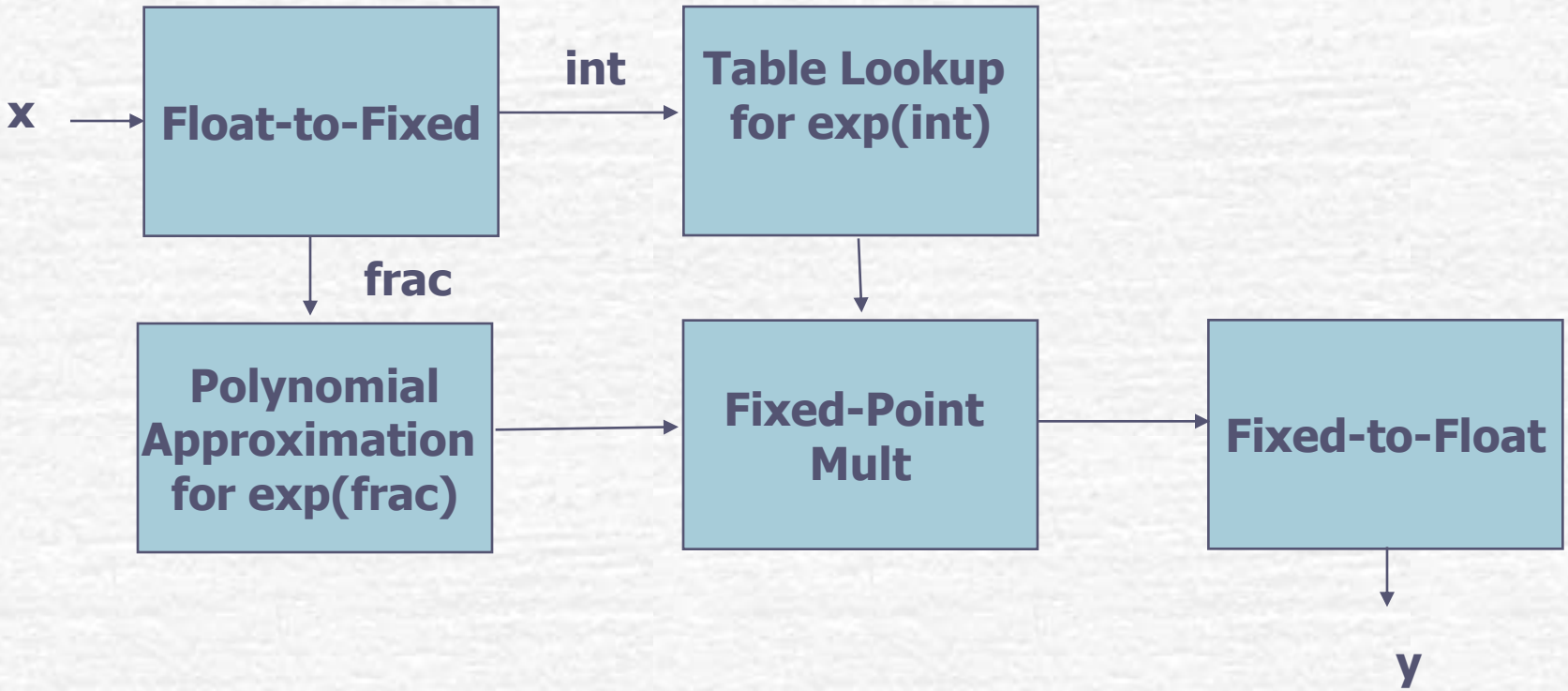
$$x = 1.f_m \ldots f_0 \times 2^{ept}$$

$$x_{fix} = fix(x) = i_p \ldots i_0 . f_q \ldots f_0$$

$$\exp(x) = \exp(x_{fix}) =$$

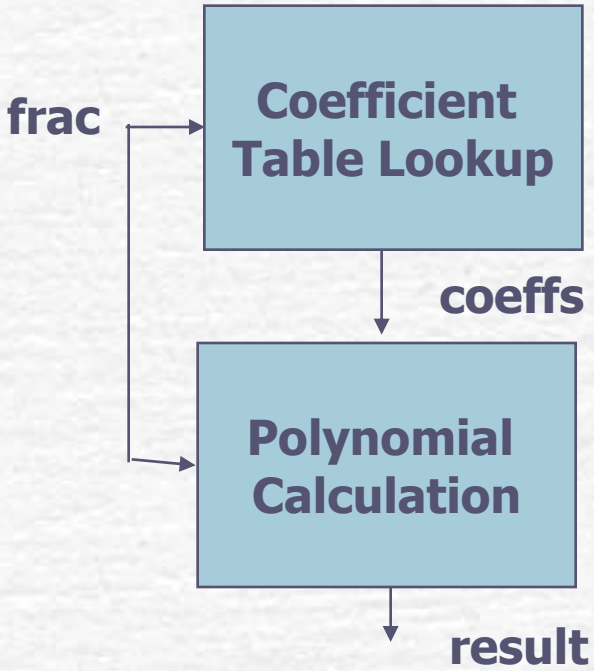
$$\exp(i_n \ldots i_0) \times \exp(0.f_m \ldots f_0)$$

Exponential Function Block Diagram



Polynomial Approximation

- Uses Interpolation of Table-Stored Values
- Degree n polynomial
- Input to Polynomial Stage Indexes Look-Up Tables
  - Look-Up in BRAMS, 256-8192 * n coeffs for d
- Usually Fixed-Point Adds and Mults



Implementation Results

Function	Slices	DSP48s	RAMB16s	Clock Rate (MHz)
tanf	10338	48	2	130.5
powf	3589	4	4	136.0
cosf	5389	24	1	168.9
sinf	3982	24	1	156.7
frexpf	26	0	0	191.2
ldexpf	44	0	0	134.5
expf	1405	0	1	150.1
logf	1736	0	3	133.5
rand_ms	150*	0	0	106.3
rand_ms_i	150*	0	0	90.4
rand	475	0	0	181.0

Speedups

Function	HW Timing (us)	SW 1 Timing (us)	SW 2 Timing (us)	HW/SW 1 Speedup	HW/SW 2 Speedup
expf	0.01	0.213	0.215	21.29	21.48
logf	0.01	0.126	0.178	12.60	17.77
sqrtf	0.01	0.199	0.246	19.92	24.61
all_funcs	0.01	0.625	N/A	62.50	N/A
PDF	0.01	0.43	N/A	43.00	N/A

Speedup Comparison Environment

- Microprocessor & ANSI-C Compiler:
  - 3.2 GHz Pentium D (dual-core) (90 nm process)
  - Windows XP
  - 2GB RAM
  - gcc –O3
- FPGA & DIME-C Compiler:
  - Virtex-4 SX35-10 FPGA (90 nm process)
  - All Logic Clocked at 100MHz
- SW1: solo function, SW2: larger function

What About Double Precision?

- Not Part of The Library At Present
- It was investigated though
- Logarithm Results Below
  - Single-Precision Techniques Scaled Poorly to Double Precision
  - Arenaire DP: 3000 slices (more on this later)

	Single	Double	Increase
Exponent Bits	8	11	1.375
Fractional Bits	24	53	2.21
Slice Consumption	1736	12000	6.91
BRAM Consumption	3	24	8.00

How Did We Implement It

- Coefficients in Mathematica
  - Little Unwieldy, Can't see under lid
- Model Architectures in C
  - Could have done with a fixed-point library
- Generate Table Data in C
- Generate VHDL components in C
- Integrate into DIME-C
- Test in hardware

What Design Techniques Did We Use

- Used VHDL, DIME-C and System Generator to Create Cores
  - No slice-to-slice comparisons
- VHDL
  - Best approach, most portable. V4 -> V5 progression
  - Pipelined Cores Bearable with Coding Discipline
- DIME-C
  - fast but required hacks, used lots more resource
- System Generator
  - Difficult to install, limited tracking of pipeline delays

What should I know about developing a core library?

- Aim for portability
  - Use VHDL/Verilog if you can
    - Better chance of successfully porting
  - Avoid instantiating hard macros if possible
    - Aim for Vendor Neutral Code
    - Hard multipliers change with generations
- Don’t over optimise
  - Use the simplest architectures and techniques that will do the job

What should I know about developing a math library?

- Learn from the French:
  - The Arenaire Project
  - Jean-Michel Muller's Book on Elementary Functions
  - Detrey & De Dinechin: *Return of the hardware floating-point elementary function*
- Can't get last-bit precision on FPGAs
  - Faithful rounding, not correct rounding is the best possible, ziv's strategy
- Double-Precision is looking more viable with new techniques.