Calculator Implementation on Vaman FPGA

Siddhant Yellanki

August 19, 2025

Outline

Introduction

1. This paper presents the design and implementation of a scientific calculator on a Vaman FPGA, using a synthesizable Verilog library for fixed-point arithmetic and elementary functions in the Q16.47 format.
2. The system includes modules for addition, multiplication, division, comparison, and transcendental functions (sine, cosine, tangent, exponential), as well as a converter for displaying fixed-point results as ASCII characters.
3. The calculator provides a user interface for entering numbers and operations, and displays results in human-readable decimal format.
4. It uses numerical methods such as Euler's method and Newton-Raphson iterations to calculate the values of the transcendental functions up to an accuracy of 4 digits.

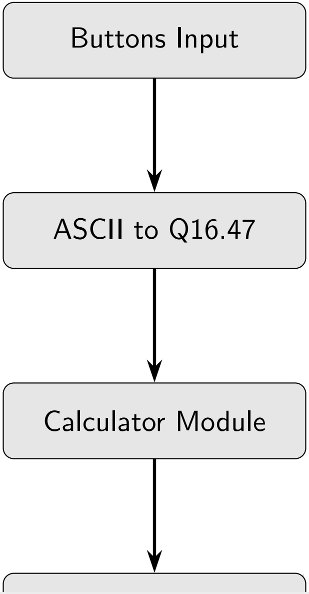
System Overview

The calculator system consists of the following components:

- User input interface (5 x 5 Button Array)
- Arithmetic and function modules (Q16.47 math library)
- ASCII conversion for display
- Output interface (16x2 LCD Display)

The entire design is implemented in Verilog and targeted for FPGA synthesis.

System Overview



Components

Component	Value	Quantity
Vaman Board		1
USB-UART		1
16x2 LCD Display		1
Push Buttons		25
Slide Switch		1
Jumper Wires	F-M	30
Wires		
Breadboard		2

Table 1.0

Input Keyboard Connections

BUTTON	VAMAN BOARD	Name
Button 1	PYGMY 1	Num 0
Button 2	PYGMY 2	Num 1
Button 3	PYGMY 3	Num 2
Button 4	PYGMY 4	Num 3
Button 5	PYGMY 5	Num 4
Button 6	PYGMY 6	Num 5
Button 7	PYGMY 7	Num 6
Button 8	PYGMY 8	Num 7
Button 9	PYGMY 9	Num 8
Button 10	PYGMY 10	Num 9

Input Keyboard Connections

Button 11	PYGMY 11	Plus
Button 12	PYGMY 12	Minus
Button 13	PYGMY 13	Multiply
Button 14	PYGMY 14	Divide
Button 15	PYGMY 15	Sin
Button 16	PYGMY 16	Cos
Button 17	PYGMY 17	Tan
Button 18	PYGMY 18	Exp
Button 19	PYGMY 19	Enter
Button 20	PYGMY 20	Decimal
Button 21	PYGMY 21	Reset
Toggle	PYGMY 22	Function Toggle

Table 2.0

LCD Connections

Make the Circuit Connections as per the table below.

Vaman	LCD	LCD Name	LCD Description
GND	1	GND	
5V	2	Vcc	
GND	3	Vee	Contrast
PYGMY 23	4	RS	Register Select
GND	5	R/W	Read/Write
PYGMY 24	6	EN	Enable
PYGMY 25	11	DB4	Serial Connection
PYGMY 26	12	DB5	Serial Connection
PYGMY 27	13	DB6	Serial Connection
PYGMY 28	14	DB7	Serial Connection
5V	15	LED+	Backlight
GND	16	LED-	Backlight

Table 3.0

Q16.47 Fixed-Point Format

A Q16.47 fixed-point number is a 64-bit signed value:

- **Bit 63:** Sign bit (0 for positive, 1 for negative)
- **Bits 62:47:** Integer part (16 bits)
- **Bits 46:0:** Fractional part (47 bits)

The value is:

$$\text{Value} = (-1)^{\text{sign}} \times \left(\text{Integer Part} + \frac{\text{Fractional Part}}{2^{47}} \right) \tag{1}$$

The reason behind this format rather than using the industry standard IEEE-754 Single Precision Floating Point format, is due to the ease of implementing, fast calculation and low error accumulation rate.

Q16.47 Fixed Point Format

1. Connections to floating point to fixed point
   Fixed point is very simple as it is simply just shifting of bits. So to convert to fixed point,

$$\mathit{Float} = \frac{\mathit{Fixed}}{2^{47}} \tag{2}$$

2. Conversion to Fixed Point from Floating Point Similarly, compared to converting to fixed point,

$$\mathit{Fixed} = \mathit{Float} \times 2^{47} \tag{3}$$

3. Range of the calculator Since, we have 63 bits of working integers, the range is $\pm 2^{63-47}$ or -65536 to 65536. In the code, 65536 is defined as infinity.

Arithmetic Modules

1. Adder
2. Multiplier
3. Divider
4. Comparison

Adder

The adder module performs signed addition of two Q16.47 numbers. This module is fully synthesizable and handles two's complement arithmetic.
Limitations and Edge Cases:

- The output saturates to the maximum if the sum exceeds the representable range of Q16.47 (i.e., overflow/underflow is possible but not flagged).

Multiplier

The multiplier module multiplies two Q16.47 numbers and aligns the result using bit shifting. This implementation uses two's complement arithmetic and checks for overflow.

Limitations and Edge Cases:

- If the product exceeds the Q16.47 representable range, the overflow flag is set.
- Overflow does not saturate the output; the result wraps around as per two's complement arithmetic.
- Multiplying by zero always yields zero.
- Multiplying the largest positive and negative values can result in overflow.

Divider

The divider module implements non-restoring division using shift and subtract. The divider supports signed division and overflow detection.

Limitations and Edge Cases:

- Division by zero: If the divisor is zero, the overflow flag is set, and the quotient output is undefined (may be set to the maximum positive or negative value, depending on implementation).
- Division overflow: If the result exceeds the Q16.47 range, the overflow flag is set.
- Division of zero by any nonzero number yields the maximum possible number 65536.

Comparison

The fixedCompare module compares the absolute values of two Q16.47 numbers. This is mainly used in the iterative approach of the transcendental functions, to compare between the current iteration and the maximum iteration.

Transcendental Function Modules

The transcendental functions are implemented using iterative or difference-equation-based numerical methods, leveraging the arithmetic modules. Below, we provide not only the update equations, but also the underlying derivations for commonly used methods such as Forward Euler, Double Euler (Leapfrog), and RK2.

sin(x): Double Euler Method

To compute $\sin(x)$, recognize the second-order ordinary differential equation:

$$\frac{d^2 y(x)}{dx^2} = -y(x)$$

where $y(x) = \sin(x)$ and $\frac{d^2 \sin(x)}{dx^2} = -\sin(x)$.
Approximating the second derivative at a point x_n using the centered finite difference:

$$\frac{y_{n+1} - 2y_n + y_{n-1}}{h^2} \approx \left.\frac{d^2 y}{dx^2}\right|_{x_n}$$

Substituting into the ODE:

sin(x):Double Euler Method

$$\frac{y_{n+1} - 2y_n + y_{n-1}}{h^2} = -y_n$$
$$\Rightarrow y_{n+1} = 2y_n - y_{n-1} - h^2 y_n$$

Initialization:

$$y_0 = 0, \quad y_1 = h, \quad h = \frac{1}{2^{16}}$$

or, in fixed point: $y_0 = 64'd0, y_1 = 64'h0000000080000000$ $,h = 64'h0000000080000000.$

For improved fixed-point precision, h may be chosen as $h = \frac{1}{2^{15}}$, meaning $h^2 = \frac{1}{2^{30}}$, which offers sufficient resolution for the system.

cos(x):Double Euler Method(different initial Condition)

Using the same update equation as for $\sin(x)$, but initializing to match the series expansion for $\cos(x)$:

$$y_0 = 1, \quad y_1 = 0.999, \quad h = \frac{1}{2^{16}}$$

or in fixed point: $y_0 = 64'h0000800000000000, y_1 = 64'h00007FFFFFFFC000, h = 64'h0000000080000000$

tan(x): Forward Euler Method

The governing ODE for the tangent is:

$$\frac{dy}{dx} = 1 + y^2$$

Applying the (explicit) forward Euler method, where the derivative is approximated as:

$$\frac{y_{n+1} - y_n}{h} \approx f(y_n, x_n)$$

leads to:

$$y_{n+1} = y_n + (1 + y_n^2)h$$

with initial conditions $y_0 = 0$, $h = \frac{1}{2^{16}}$, or in fixed point, $y_0 = 64'd0$, $h = 64'h0000000080000000$.

exp(x): RK2 (Midpoint Method)

For exponentials, the differential equation is:

$$\frac{dy}{dx} = y$$

To solve with second-order accuracy, the RK2 (midpoint) method is:

$$k_1 = hf(x_n, y_n) = hy_n$$
$$k_2 = hf\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right) = h(y_n + \frac{k_1}{2})$$
$$y_{n+1} = y_n + \frac{1}{2}(k_1 + k_2)$$

exp(x): RK2 (Midpoint Method)

This is derived by taking an Euler step to the midpoint, evaluating the slope there, and then updating from the average of the starting and midpoint slopes.
Initial conditions: $y_0 = 1$, $x_0 = 0$, $h = 0.01$.
In fixed point:
$x_0 = 64'd0$, $y_0 = 64'h0000800000000000$, $h = 64'h0000000080000000$.

Summary Table of Methods

Function	ODE	Method	Iteration Formula
$\sin(x)$	$y''(x) = -y(x)$	Double Euler	$y_{n+1} = 2y_n - y_{n-1} - h^2 y_n$
$\cos(x)$	$y''(x) = -y(x)$	Double Euler	$y_{n+1} = 2y_n - y_{n-1} - h^2 y_n$
$\tan(x)$	$y'(x) = 1 + y^2$	Euler	$y_{n+1} = y_n + (1 + y_n^2)h$
$\exp(x)$	$y'(x) = y(x)$	RK2	$k_1 = hy_n$ $k_2 = h(y_n + \frac{k_1}{2})$ $y_{n+1} = y_n + \frac{k_1 + k_2}{2}$

The fta Module: Fixed-Point to ASCII Conversion

The fta module converts a fixed-point binary number into its human-readable ASCII decimal representation. This is essential for displaying calculation results on an LCD or other character-based interfaces.

- **Input:** A fixed-point number value of width N (e.g., 64 bits), with Q bits for the fractional part.
- **Output:** An array of ASCII characters (ascii_array) representing the signed decimal value, e.g., "-12.3456".

The `fta` Module: Fixed-Point to ASCII Conversion

The conversion process:

1. **Sign Extraction:** The sign bit is checked. If negative, the value is negated and a "-" is added to the ASCII output.
2. **Integer Part:** The integer portion is extracted by right-shifting the value by Q bits. This integer is then converted to ASCII digits by repeated division and modulo by 10.
3. **Decimal Point:** A "." character is inserted after the integer digits.
4. **Fractional Part:** The fractional portion (the lower Q bits) is scaled up (multiplied by 10^n for n decimal digits) and right-shifted by Q to get the decimal representation, which is then converted to ASCII digits.
5. **Output Formatting:** The ASCII characters are stored in the output array, padded with spaces if necessary.

The `atf` Module: ASCII to Fixed-Point Conversion

The `atf` module performs the reverse operation: converting an ASCII string representing a decimal number into a fixed-point binary value.

- **Input:** An array of ASCII characters (`ascii_array`) representing a number, e.g., “-12.3456”.
- **Output:** A fixed-point number `value` of width N and Q fractional bits.

The fta Module: Fixed-Point to ASCII Conversion

The conversion process:

1. **Sign Detection:** Checks for a leading "-" character to determine if the result should be negative.
2. **Integer Part Parsing:** Reads ASCII digits before the decimal point, accumulating the integer value.
3. **Fractional Part Parsing:** After the decimal point, reads ASCII digits, accumulating the fractional value and counting the number of digits.
4. **Fixed-Point Assembly:** The integer part is left-shifted by Q bits. The fractional part is scaled to fit into Q bits (i.e., `frac_accum * 2Q̂ / 10d̂`, where d is the number of fractional digits).
5. **Sign Application:** If the input was negative, the two's complement is applied to the result.

Performance

All modules use synthesizable constructs:

- **Combinational logic:** Implemented with `always @(*)`.
- **Static arrays and loops:** Loops are unrolled by synthesis tools.
- **Arithmetic:** Addition, multiplication, and division are synthesizable, though division by 10 (for ASCII conversion) may be resource-intensive.
- **Parameterization:** All modules are parameterized for flexibility.

Overflow detection and sign handling are included throughout.

Accuracy

To measure the accuracy of the calculator, we will compare it with C standard results.

1. Sin(x)
2. cos(x)
3. tan(x)
4. exp(x)

Sin(x)

x	$sinFixed(x)$	$sin(x)$
1	0.8414792291	0.8414709848
2	0.9092910765	0.9092974268
1.57	0.9999997006	0.9999996829

Cos(x)

x	$cosFixed(x)$	$cos(x)$
1	0.5402894659	0.5403023059
2	-0.4161607117	-0.4161468365
1.57	0.0007737434	0.0007963267

Tan(x)

x	$tanFixed(x)$	$tan(x)$
1	1.5573755479	1.5574077247
1.57	1186.2765882125	1255.7655915008

Exp(x)

x	$expFixed(x)$	$exp(x)$
1	2.7182714591	2.7182818285
1.5	4.4816634261	4.4816890703
3.2	24.5325301975	24.5325301971

Execution

- Download the repository

```
git clone https://github.com/ysiddhanth/vaman.git
cd vaman
```

- Locate the folder codes, in the Calculator folder.

```
cd Calculator/codes
```

- Generate the .bin file

```
ql_symbiflow -compile -src . -d ql-eos-s3 -P pu64 -t main
```

- Dump .bin files on the Vaman FPGA

```
sudo python3 <tinyfpga uploader> --port /dev/ttyACM0  --ap
```

Hardware Built

- Connect the LCD Display and the buttons to the breadboard and make the connections to the fpga according to the above tables.

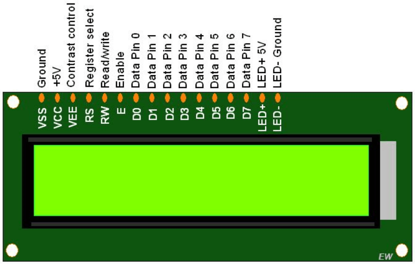


Figure 1 - LCD Pinout

Hardware Built

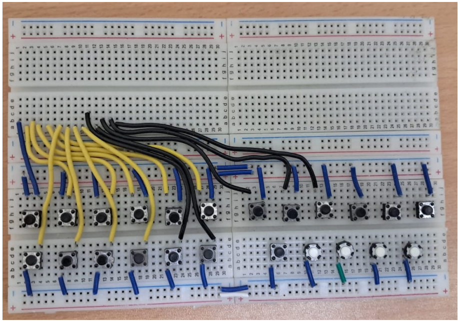


Figure 2 - Button Arrangement

Hardware Built

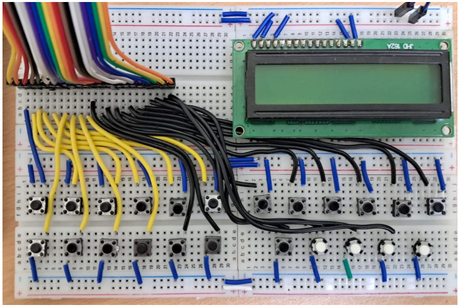


Figure 3 - The calculator