분류번호 : 2020-2-4108-03

Design of 32-bit FPU by Posit Number Format and Application in Embedded Systems

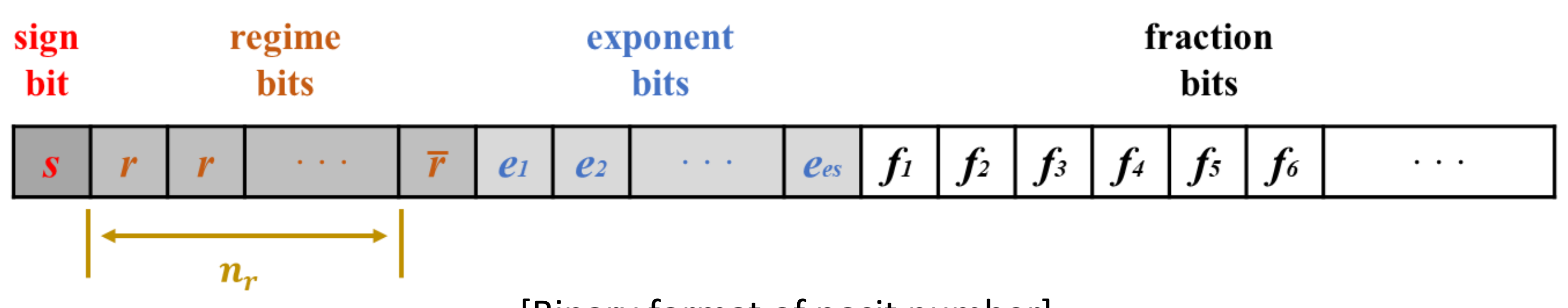
Hyun Woo Oh, Seung Hun Kim and Seung Eun Lee

Dept. of Electronic and IT Media Engineering
Seoul National University of Science and Technology

Abstract

Nowadays, floating-point arithmetic has been a big part of many areas such as machine learning and 3D graphics. IEEE 754 is most common in floating-point computations. However, IEEE 754 has a limitation of range and precision because of the fixed length of exponent bits and fraction bits. Posit is an alternative to IEEE 754 to deal with these limitations. Posit, unlike IEEE 754, has variable width of exponent bits and fraction bits. In this work, we designed the arithmetic unit that operates calculation between 32-bit posit numbers, and the coprocessor of MIPS processors to be compatible with MIPS's floating-point instructions. Finally, we demonstrated our design on a field-programmable gate array(FPGA).

Posit Number



[Binary format of posit number]

$$N = -1^{s} \times 2^{2^{es}\times k+e} \times f$$

[Number expression of posit number system]

s	:	Sign bit (0 or 1)
k	:	$k = \begin{cases} k = -1 \times n_r & (r=0) \\ k = n_r - 1 & (r=1) \end{cases}, \quad n_r = bitwidth\ of\ r$
es	:	Width of the exponent bits.
e	:	Number value of exponent bits. $\quad e = 0 \sim 2^{es} - 1$
f	:	Fraction bits. $\quad f = 1.f_1 f_2 f_3 f_4 \cdots_{(2)}$

range of posit (32-bit, es=3)	$: -2^{248} \sim 2^{240}$
precision of posit (32-bit, es=3)	$: 0 \sim 26\ bits$

MIPS Floating-Point Instructions

Arithmetic
- ADD
- SUB
- MUL
- DIV
- ABS
- NEG
- SQRT
- MOV

Conversion
- CVT.S
- CVT.W

Compare
- C.cond.S

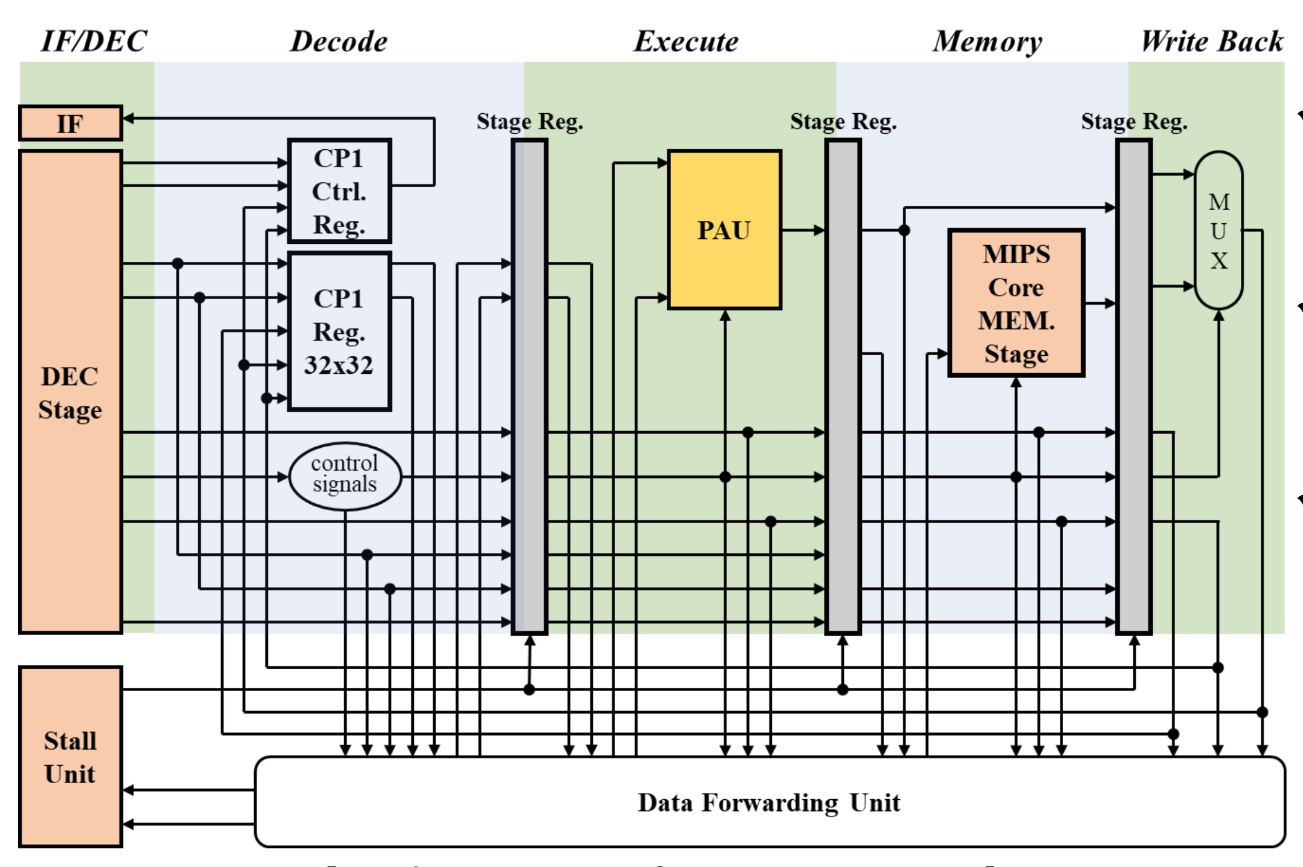
Branch
- BC1T
- BC1F

Move
- MFC1
- MTC1
- CFC1
- CTC1

Load & Store
- LWC1
- SWC1

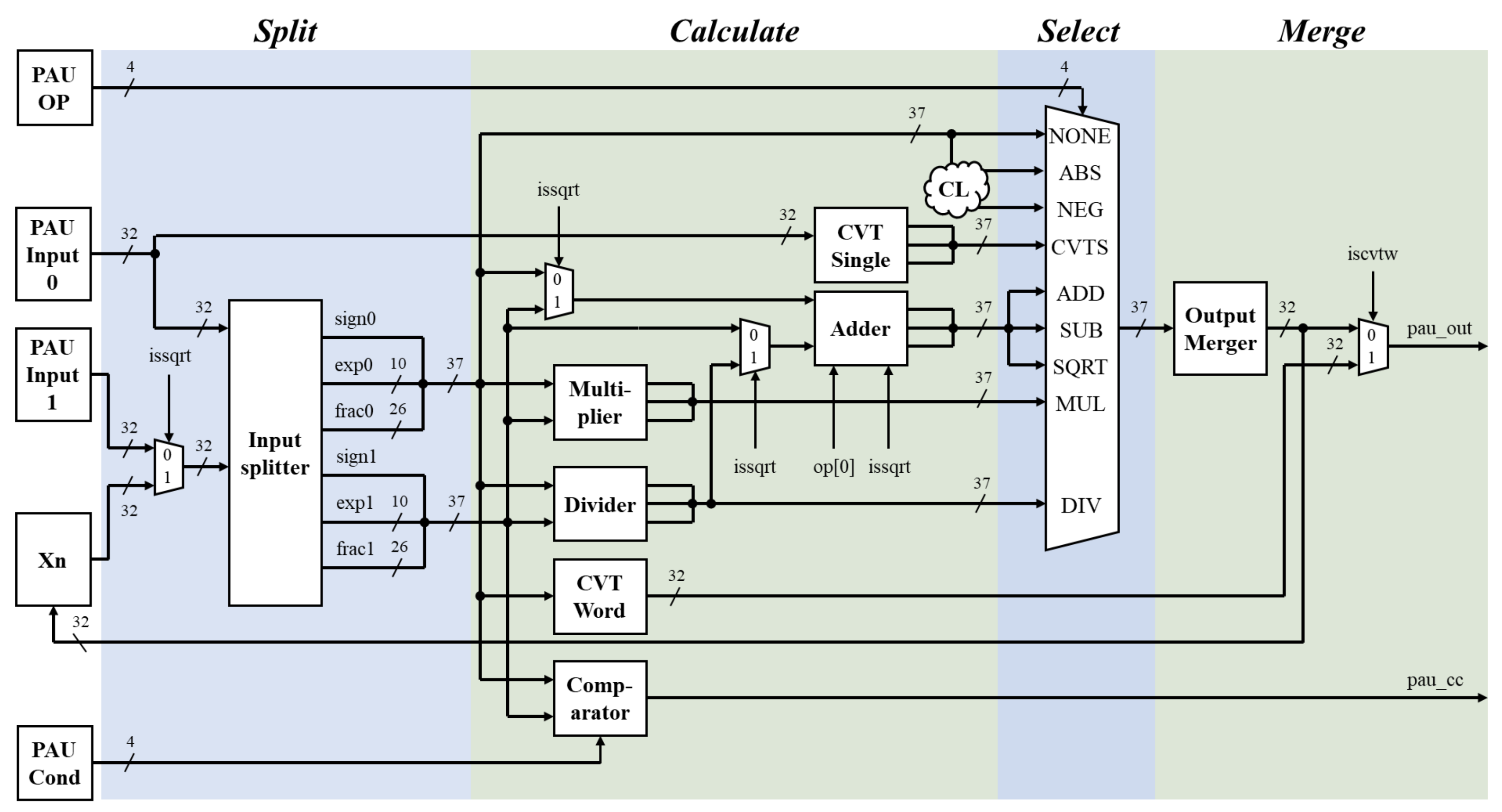
🟦 PAU
🟪 CP1
🟧 Core MEM

System Architecture



[Architecture of Coprocessor 1]

- ✓ Five pipelined stages IF, DEC, EX, MEM, WB
- ✓ Shares IF, DEC stage with MIPS core
- ✓ Memory access runs through the MIPS core



[Posit arithmetic unit architecture]

PAU consists of four stages called **Split**, **Calculate**, **Select**, and **Merge**.

Split
- Decodes each two posit (32-bit, es=3) inputs to sign bit, 10 exponent bits, and 26 fraction bits, a total of 37 bits.
- Passes decoded data to each **Calculate** stage modules.

Calculate
- Computes from input data and passes 37-bit result to **Select** stage.
- SQRT computes by iterating four times of the formula below.

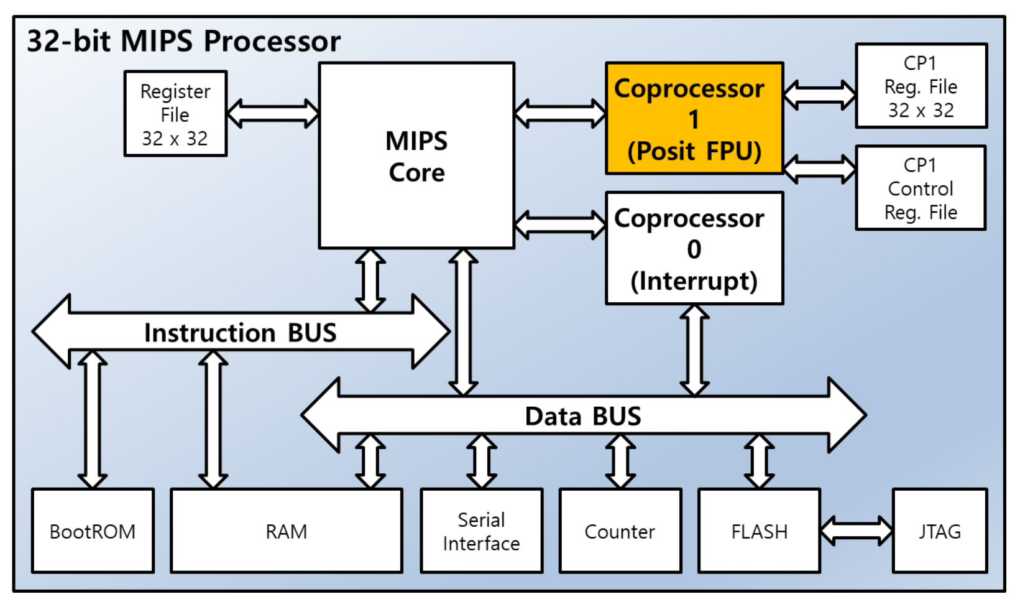
$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right), \qquad \begin{cases} a = input \\ x_1 = 1 \end{cases}$$

Select
- Selects the result from the **Calculate** stage modules that matches the current instruction code and passes result to **Merge** stage.

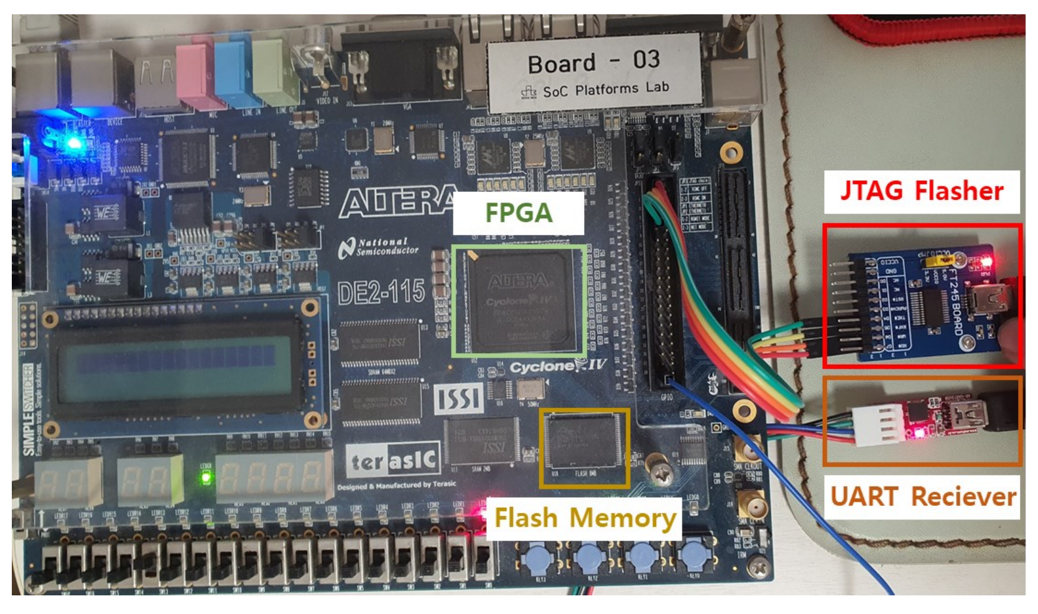
Merge
- Encodes the 37-bit data from **Select** stage to posit (32-bit, es=3).
- Selects and Outputs the correct data from posit data and integer data.

Demonstration

Demonstration Flow
- Compile and build test program through mips-elf-gcc compiler
- Upload compiled binary to FLASH memory of FPGA board through JTAG and PC upload program
- Check the result of the test program through a serial interface and PC

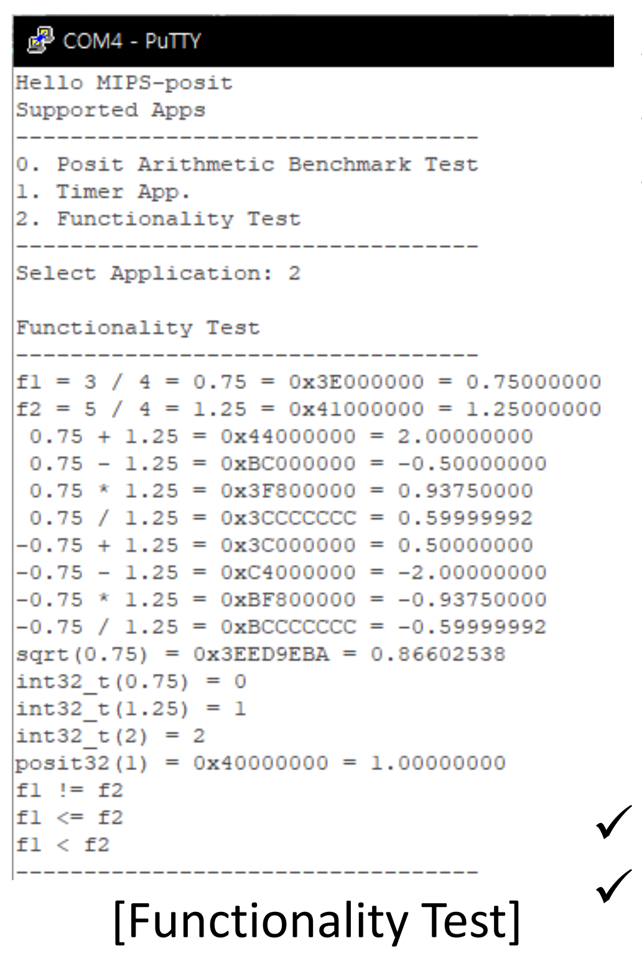


[Entire architecture of processor]



[Verification environment]

Test Program

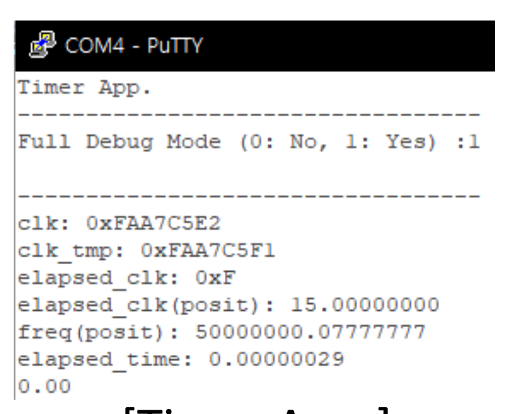


```
COM4 - PuTTY
Hello MIPS-posit
Supported Apps
--------------------------------------
0. Posit Arithmetic Benchmark Test
1. Timer App.
2. Functionality Test
--------------------------------------
Select Application: 2

Functionality Test
--------------------------------------
f1 = 3 / 4 = 0.75 = 0x3E000000 = 0.75000000
f2 = 5 / 4 = 1.25 = 0x41000000 = 1.25000000
 0.75 + 1.25 = 0x44000000 = 2.00000000
 0.75 - 1.25 = 0xBC000000 = -0.50000000
 0.75 * 1.25 = 0x3F800000 = 0.93750000
 0.75 / 1.25 = 0x3CCCCCCC = 0.59999992
-0.75 + 1.25 = 0x3C000000 = 0.50000000
-0.75 - 1.25 = 0xC4000000 = -2.00000000
-0.75 * 1.25 = 0xBF800000 = -0.93750000
-0.75 / 1.25 = 0xBCCCCCCC = -0.59999992
sqrt(0.75) = 0x3EED9EBA = 0.86602538
int32_t(0.75) = 0
int32_t(1.25) = 1
int32_t(2) = 2
posit32(1) = 0x40000000 = 1.00000000
f1 != f2
f1 <= f2
f1 < f2
--------------------------------------
```

[Functionality Test]

- ✓ Runs the codes including FPU instructions
- ✓ Runs through posit arithmetic
- ✓ Shows the result of the majority of FPU instructions by hexadecimal and real representation

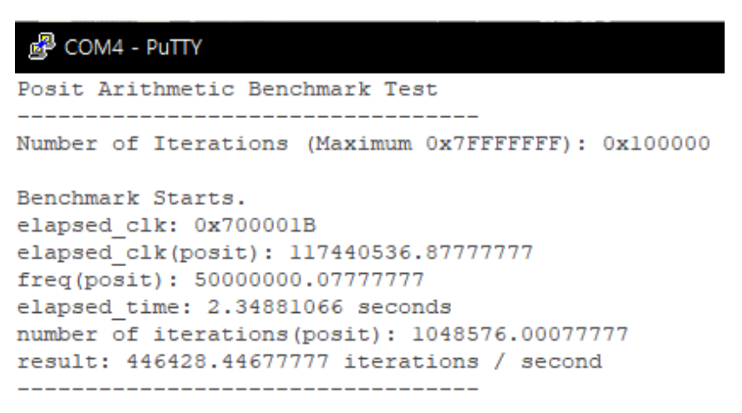


```
COM4 - PuTTY
Timer App.
--------------------------------------
Full Debug Mode (0: No, 1: Yes) :1

--------------------------------------
clk: 0xFAA7C5E2
clk_tmp: 0xFAA7C5F1
elapsed_clk: 0xF
elapsed_clk(posit): 15.00000000
freq(posit): 50000000.07777777
elapsed_time: 0.00000029
0.00
```

[Timer App.]

- ✓ Simple timer
- ✓ Runs through posit arithmetic from FPU instructions
- ✓ Operates by counter peripheral



```
COM4 - PuTTY
Posit Arithmetic Benchmark Test
--------------------------------------
Number of Iterations (Maximum 0x7FFFFFFF): 0x100000

Benchmark Starts.
elapsed_clk: 0x700001B
elapsed_clk(posit): 117440536.87777777
freq(posit): 50000000.07777777
elapsed_time: 2.34881066 seconds
number of iterations(posit): 1048576.00077777
result: 446428.44677777 iterations / second
--------------------------------------
```

[Benchmark Test]

- ✓ Set the number of times to iterate
- ✓ Iterates the specific codes
- ✓ Checks the execution time