Hardware-Based Algorithm for Sine and Cosine Computations using Fixed Point Processor

My X. NGUYEN
Ho Chi Minh City University of Technology
Ho Chi Minh City, Viet Nam
xuanmy92@gmail.com

Anh-Vu DINH-DUC
University of Information Technology-VNUHCM
Ho Chi Minh City, Viet Nam
anhvu@uit.edu.vn

Abstract—**This paper presents an algorithm for computing Sine and Cosine functions. The algorithm is developed for Fixed Point Processor. It only required fixed point multiplier 32x32 and fixed point adder. In addition, the algorithm uses Table Lookup method with the size of table being less than 2 Kbytes. Hence, computations are speed up significantly although it does not require many resources. One of the advantages points is that the algorithm computes Sine and Cosine functions without extra floating point units; it can compute fraction numbers under fixed point format. In some applications which require high accuracy in computing, this algorithm needs to be analyzed carefully to satisfy the error in computing. The algorithm's correction is verified successfully by ModelSim-Altera software.**

Keywords— Table-base algorithm; Polynomial Approximation algorithm; trigonometric functions; sine cosine functions; fixed point computing; digital arithmetic hardware.

I. Introduction

Trigonometric function is one of the most important elementary functions in engineering technology. Algorithm for computing this function was discovered early and CORDIC (COordinate Rotation DIgital Computer) is one of the successful algorithms to solve the problem. This algorithm is discovered by Volder [4] and unified by Walther [6]. It is commonly used when hardware does not have multiplier. However, calculating speed of the CORDIC algorithm is not satisfactory in large scale computing, or particularly numerically intensive computations. In supercomputer, CORDIC is not willing to the significant use of pipelining. Other method based on polynomial approximation [3], interpolation or rational functions [7] have become the model in computing elementary function on such machine. In order to attain high speed in all of these methods, the pre-computed tables of value are applied. With modern technology, the speed and density of memory increase drastically, table-based methods are developed easily. All fast methods for computing elementary functions [1], [5], [8] use tables.

From modern results in ASIC field [13], we would like to build independent components which can be embedded into DSP cores. These components help DSP system to be developed easily and be more powerful.

This paper will describe new table-based algorithm for computing Sine-Cosine function for fixed point number with specific format. This algorithm only requires fixed point processor and high efficiency when multiplier is already available in the processor. Sine-Cosine function is one of the most important elementary functions in scientific, hence it warrants the extra investment in hardware. We hope that this algorithm will increase speed in computing scientific applications.

The rest of this paper is constructed as follow: Section 2 is for presenting techniques for computing Sine-Cosine in hardware. The algorithm details will be described in Section 3. And the last section is for results, summary and the future works.

This work was supported in part by the Department of Science & Technology, Ho Chi Minh City, Viet Nam.

II. Approaches for Computing Sine-Cosine in Hardware

This section will describe some well-know types of techniques for computing transcendental function in hardware. In order to focus on problem of this paper, we will only mention methods which relate to our algorithm.

A. *Table lookup method.*

The table lookup method uses the table to store outputs of the function to be computed for every argument *x*. The number of table entries is depended on the number of bits of the representation of the argument. It will be exponential in number of bits of the argument, e.g.: function *f* has an argument *x* with 8 bits representation, then the table involves 256 entries. This method speeds up computing in runtime. However, it also consumes a huge resource if we have the argument with many bits of the representation.

B. *Polynomial approximation method.*

This method bases on representation of a function as an infinite sum of terms. The number of term which needs to be computed is depending on our accuracy in computing functions. For example, representation of Sine-Cosine functions as Taylor's series:

$$\sin x = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots, \ -\infty < x < \infty \qquad (1)$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots, \ -\infty < x < \infty \qquad (2)$$

We call ε is error of final result in computing function (1) and (2). Assumption:

- Required error of final result: $\varepsilon < 2^{-31}$.

- We use temporary registers r to store value of terms and $r > 2^{-31}$

Then, with input $x < 2^{-7}$, only two terms will be computed for each function as below:

$$\sin x = \frac{x}{1!} - \frac{x^3}{3!} \quad (3)$$

$$\cos x = 1 - \frac{x^2}{2!} \quad (4)$$

Because of registers r can not store value of $\frac{2^{-7*5}}{5!}$ and value of $\frac{2^{-7*4}}{4!}$, as above assumption $r > 2^{-31}$.

We realize that many terms of polynomial will be eliminated when computing functions which base on Taylor's series with inputs are near zero.

C. Combined methods.

Above methods have disadvantages if we use them separately. About Table Lookup method, with argument has many bits we need a large table for storing entries, e.g.: it is too waste to create a table contains 2^{32} entries in hardware when we apply this method for an argument with 32 bits in representation. About Polynomial Approximation method, many terms with many multiplications need to be computed when inputs are not near zero. It will take more time to attain acceptable accuracy in computing our functions. In other to balance memory and speed and to reduce disadvantage points of these two methods we will combine the above two methods. For reference about this combination see [2], [9], [10] and [11].

III. Simple Sine-Cosine Computing Algorithm

Before deeply digging on algorithm, there are some points which need to be noted. As explained of Koren's text [12], we can limit our domain of input to $\left[0, \frac{\pi}{2}\right]$ for Sine and Cosine functions. With this domain, results of Sine-Cosine functions are always in value range [0, 1]. Our algorithm is applied for fixed point processor, hence we must find out the way to represent a fraction number under fixed point format. Assumption we call it as M format, this format will be described more details as below. One more point need to be noted that this algorithm is developed for fixed point 32-bit processor using fixed point multiplier 32x32 and 32-bit registers.

A. M format

M format is a kind of fixed point number format where the number of fractional bits is specified. As above, we will use 32-bit register, so we assume that inputs for Sine-Cosine functions have 32 bits. M format need to satisfy two conditions: first it must cover input domains $\left[0, \frac{\pi}{2}\right]$ (input domains already cover output domains [0, 1]), second it must achieve high accuracy as possible. In order to reach two conditions we decide that from 32 bits there is one bit for representing integer value (most significant bit) and remained 31 bits for representing fractional value. And *dot* symbol which is used to determine integer value and fractional value is to stand between bit 31 and bit 30 as below picture.

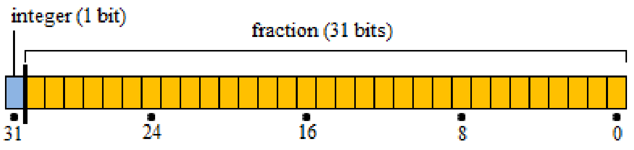


Figure 1 - M format

With above M format, we have the formula to calculate value of number n which is represented as fixed point 32-bit format.

$$n = \text{bit}_{31}*2^0 + \text{bit}_{30}*2^{-1} + \text{bit}_{29}*2^{-2} + \text{bit}_{28}*2^{-3} + \ldots + \text{bit}_1*2^{-30} + \text{bit}_0*2^{-31} \quad (5)$$

Some examples of M format representation.

TABLE I. M FORMAT REPRESENTATION AND VALUE

Fixed point representation (32 bits)	*Value*
0000 0000 0000 0000 0000 0000 0000 0000[a]	0.0
0000 0000 0000 0000 0000 0000 0000 0001	$2^{-31} \approx 4.66 \times 10^{-10}$
0000 0000 0000 0000 0000 0000 0000 0010	$2^{-30} \approx 9.31 \times 10^{-10}$
.	.
.	.
.	.
0111 1111 1111 1111 1111 1111 1111 1111	$1 - 2^{-31} \approx 0.99999999953$
1000 0000 0000 0000 0000 0000 0000 0000	$2^0 = 1$
1000 0000 0000 0000 0000 0000 0000 0001	$2^0 + 2^{-31} \approx 1.00000000047$
.	.
.	.
.	.
1111 1111 1111 1111 1111 1111 1111 1101	$2 - 2^{-31} - 2^{-30} \approx 1.99999999860$
1111 1111 1111 1111 1111 1111 1111 1110	$2 - 2^{-31} - 2^{-31} \approx 1.99999999907$
1111 1111 1111 1111 1111 1111 1111 1111	$2 - 2^{-31} \approx 1.99999999953$

As above table, M format can represent numbers with its value is in value range $[0, 2 - 2^{-31}]$ under 32-bit fixed point format. Evidently, this range totally covers range $\left[0, \frac{\pi}{2}\right]$. M format has constant resolution is 2^{-31}.

With representing fractional numbers under fixed point format, calculating of fractional numbers can use fixed point multiplier or adder. It will increase computing speed many times when comparing floating point computer for computing fractional numbers.

B. Idea of algorithm

Our idea is splitting an input into two inputs, and then using the technique: combining Table Lookup method and Polynomial Approximation method to compute Sine-Cosine functions. Assumption: we have input n and it is represented by M format, n is split into a and b like that: $n = a + b$. Then, using well-known trigonometric functions, sin n and cos n will be converted as below:

$\sin n = \sin(a + b) = \sin a * \cos b + \cos a * \sin b$ (6)

$\cos n = \cos(a + b) = \cos a * \cos b - \sin a * \sin b$ (7)

With:

$$a \in \left[2^{-7}, \frac{\pi}{2}\right]$$

$$b \in [0, 2^{-7})$$

Or, under bit representation with M format:

$n = b_{31}.b_{30}b_{29}b_{28}\_b_{27}b_{26}b_{25}b_{24}\_\ b_{23}b_{22}\ldots\ b_5b_4\_b_3b_2b_1b_0$ (8)

$a = b_{31}.b_{30}b_{29}b_{28}\_b_{27}b_{26}b_{25}b_{24}\_00\ldots00\_0000$ (9)

$b = 0.000\_0000\_b_{23}b_{22}b_{21}b_{20}\_b_{19}b_{18}\ldots b_5b_4\_\ b_3b_2b_1b_0$ (10)

For explaining reason to split a, b as above refer (13).

Now, we will compute sin a, cos a and sin b, cos b.

1) Computing sin a, cos a:

We realize that there are only 8 bits of a which affect to result, these bits are from bit at position 24 to bit at position 31. Hence it is possible to applying Table Lookup method for computing sin a and cos a. Our table take 8-bit input, each entry stored result of sin a or cos a and the result also is under M format. There are two tables, one for Sine function and remained one for Cosine function. Some sample entries for Sine-Cosine functions:

TABLE II. VALUES OF SIN A

M format of a (only represent from bit 24 to bit 31)	*Value of a*	*M format of sin a*	*Value of sin a*
0.000 0000	0	0.000 0000 0000 0000 0000 0000 0000 0000	0.0
0.000 0001	0.0078125	0.000 0000 1111 1111 1111 1111 0101 0101	≈ 0.0078124053
0.000 0010	0.015625	0.000 0001 1111 1111 1111 1010 1010 1010	≈ 0.0156243639
⋮		⋮	⋮
1.100 0111	1.5546875	0.111 1111 1111 1011 1011 1111 1010 0000	≈ 0.9998702556
1.100 1000	1.5625	0.111 1111 1111 1110 1101 1111 0100 1111	≈ 0.9999655853
1.100 1001	1.5703125	0.111 1111 1111 1111 1111 1111 0000 0100	≈ 0.9999998827

TABLE III. VALUES OF COS A

M format of a (only represent from bit 24 to bit 31)	*Value of a*	*M format of cos a*	*Value of cos a*
0.000 0000	0	1.000 0000 0000 0000 0000 0000 0000 0000	1.0
0.000 0001	0.0078125	0.111 1111 1111 1111 0000 0000 0000 0000	≈ 0.9999694824
0.000 0010	0.015625	0.111 1111 1111 1100 0000 0000 0000 0101	≈ 0.999877932
⋮		⋮	⋮
1.100 0111	1.5546875	0.000 0010 0000 1111 1101 0100 1100 1010	≈ 0.0161081301
1.100 1000	1.5625	0.000 0001 0000 1111 1101 1001 1101 0101	≈ 0.0082962313
1.100 1001	1.5703125	0.000 0000 0000 1111 1101 1010 1010 0010	≈ 0.0004838267

Notes: "dot" symbol in bits chain just makes readable Q format, no other meaning.

In above two tables, the table end at bit chain '1100 1001' of a, not '1111 1111'. We realize that if bit chain $b_{31}b_{30}\ldots b_{24}$ of a is '1100 1010' (corresponding a = 1100_1010_000_0000_0000_0000_0000_0000 = 1.578125) then $a > \frac{\pi}{2}$ and a is out of our domain $\left[0, \frac{\pi}{2}\right]$. Hence, any bit chains which are greater than '1100 1001' are redundant and do not need to store them in the table. From this reason, each our table will have 201 entries.

2) Computing cos b, sin b:

Our algorithm is developed for 32-bit processor, and using 32-bit register to store value in computing. As analysis in above Polynomial Approximation method and refer (1), (2), (3) and (4). Then:

$$\sin b = \frac{b}{1!} - \frac{b^3}{3!} \quad (11)$$

$$\cos b = 1 - \frac{b^2}{2!} \quad (12)$$

From (6), (7), (11), (12) and Table II, Table III, we can compute result of sin n and cos n.

Explanation for condition of a, b: (13)

We assume that r is temporary register to store result of terms in Taylor's series, r is 32-bit register and follow M format. Hence, register r can not store any values are less than 2^{-31}. In other to balance resource and computing speed a, b are split as (9) and (10). With this splitting, we need two tables, each table has 201 entries, and we must compute terms: $\frac{b^2}{2!}$ and $\frac{b^3}{3!}$.

Let's assume that we want to speed up computing by eliminating a term such as $\frac{b^3}{3!}$. This term is only ignored when its result is less than 2^{-31}. Because register r which is used to store result of term $\frac{b^3}{3!}$ ($r = \frac{b^3}{3!}$) can not store any values are less than 2^{-31}. Then: $\frac{b^3}{3!} < 2^{-31}$ or $b < 2^{-10}$

$b < 2^{-10}$ if and only if all bits of b from position 21 to position 31 are clear, in other word b has format as below:

$b = 0.000\_0000\_000b_{20}\_b_{19}b_{18}\ldots b_5b_4\_\ b_3b_2b_1b_0$ (14)

This leads a have 11 bits affect to result of sin a and cos a, it means that number of entries in each our table will increase 8 times. We need more resource for table, this is not necessary to consume many resources to reduce a small computing.

In case we want to reduce size of table, e.g.: a has only 7 bits which affect to result of sin a and cos a, each our table can reduce around 100 entries. And each result of sin a or cos a take 4 bytes, memory will be reduced around 400 bytes, it is not significant difference. In addition, we must compute extra terms such as $\frac{x^4}{4!}$.

C. *Steps of algorithm*

This section will describe steps to execute the algorithm.

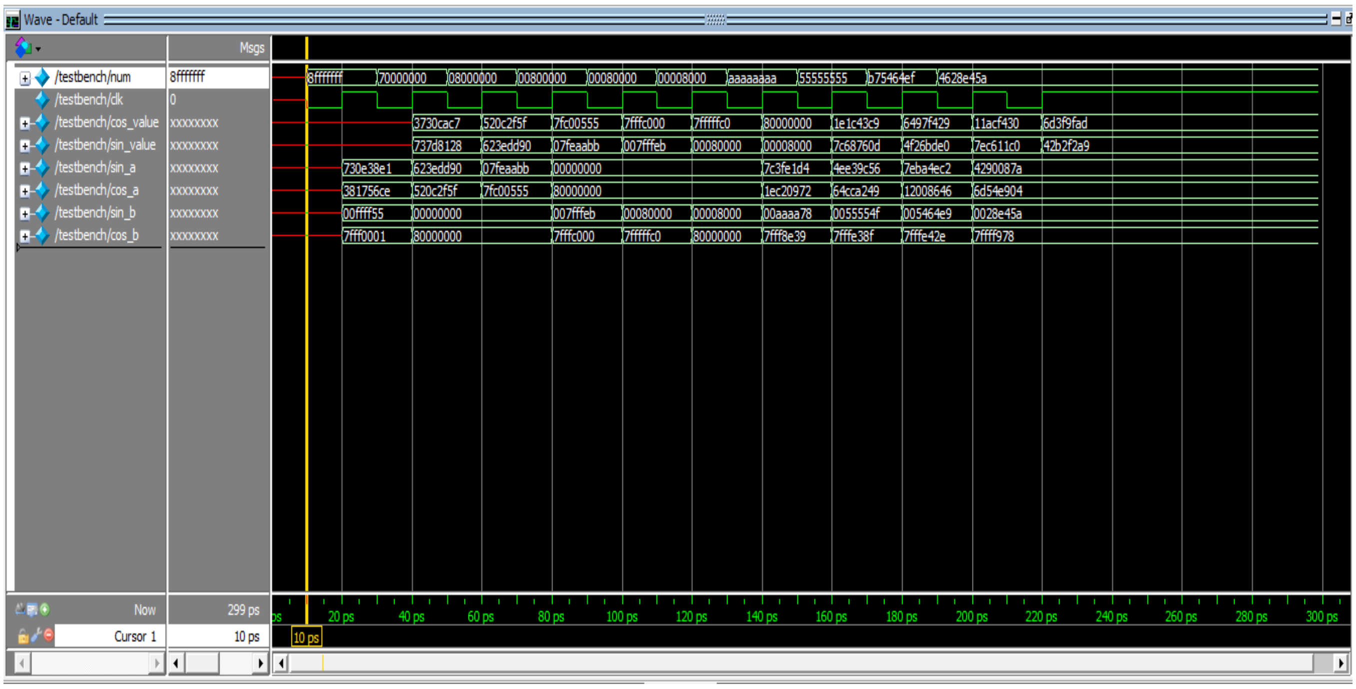
INPUT: n is a 32-bit variable under M format, and a indicator flag to determine which function need to be computed: Sine or Cosine.

- Step 1: Comparing n with $\frac{\pi}{2}$, if $n = \frac{\pi}{2}$ then base on indicator flag result will be '0' or '1' and end computing. If not equal go to Step 2.
- Step 2: From 32 bits of n, we obtain 8 high-order bits (from bit at position 24 to bit at position 31) And store them into register t. Remained bits are stored by b.
- Step 3: Obtain value of sin a, cos a from table corresponding value of t. Value of sin a, cos a is stored in two registers: r_{sa} = sin a and r_{ca} = cos a.
- Step 4: Compute sin b, cos b and store them into r_{sb}, r_{cb}.

$$r_{sb} = \sin b = \frac{b}{1!} - \frac{b^3}{3!}$$

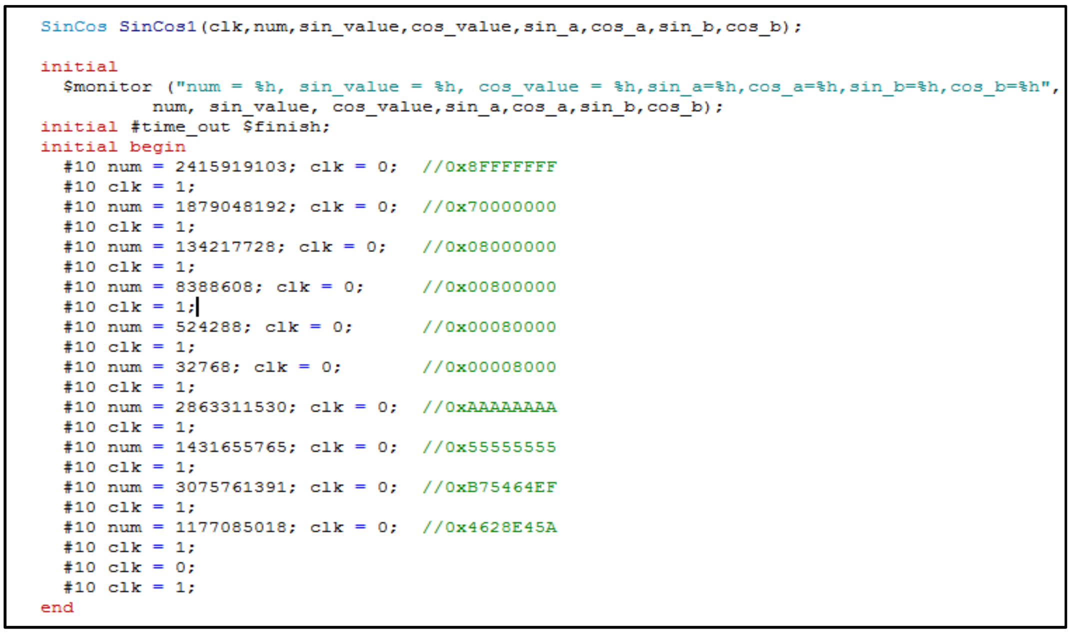
$$r_{cb} = \cos b = 1 - \frac{b^2}{2!}$$

- Step 5: Base on the indicator flag, final result is computed for Sine or Cosine functions.



```
/testbench/num        8fffffff   8fffffff 70000000 08000000 00800000 00080000 00008000 aaaaaaaa 55555555 b75464ef 4628e45a
/testbench/clk        0
/testbench/cos_value  xxxxxxxx   3730cac7 520c2f5f 7fc00555 7fffc000 7ffffc0 80000000 1e1c43c9 6497f429 11acf430 6d3f9fad
/testbench/sin_value  xxxxxxxx   737d8128 623edd90 07feaabb 007fffeb 00080000 00008000 7c68760d 4f26bde0 7ec611c0 42b2f2a9
/testbench/sin_a      xxxxxxxx   730e38e1 623edd90 07feaabb 00000000 7c3fe1d4 4ee39c56 7eba4ec2 4290087a
/testbench/cos_a      xxxxxxxx   381756ce 520c2f5f 7fc00555 80000000 1ec20972 64cca249 12008646 6d54e904
/testbench/sin_b      xxxxxxxx   00ffff55 00000000 007fffeb 00080000 00008000 00aaaa78 0055554f 005464e9 0028e45a
/testbench/cos_b      xxxxxxxx   7fff0001 80000000 7fffc000 7ffffc0 80000000 7fff8e39 7fffe38f 7fffe42e 7ffff978
```

Figure 2 - Some test-bench results on Modelsim tool



```
SinCos SinCos1(clk,num,sin_value,cos_value,sin_a,cos_a,sin_b,cos_b);

initial
  $monitor ("num = %h, sin_value = %h, cos_value = %h,sin_a=%h,cos_a=%h,sin_b=%h,cos_b=%h",
          num, sin_value, cos_value,sin_a,cos_a,sin_b,cos_b);
initial #time_out $finish;
initial begin
  #10 num = 2415919103; clk = 0;  //0x8FFFFFFF
  #10 clk = 1;
  #10 num = 1879048192; clk = 0;  //0x70000000
  #10 clk = 1;
  #10 num = 134217728; clk = 0;   //0x08000000
  #10 clk = 1;
  #10 num = 8388608; clk = 0;     //0x00800000
  #10 clk = 1;
  #10 num = 524288; clk = 0;      //0x00080000
  #10 clk = 1;
  #10 num = 32768; clk = 0;       //0x00008000
  #10 clk = 1;
  #10 num = 2863311530; clk = 0;  //0xAAAAAAAA
  #10 clk = 1;
  #10 num = 1431655765; clk = 0;  //0x55555555
  #10 clk = 1;
  #10 num = 3075761391; clk = 0;  //0xB75464EF
  #10 clk = 1;
  #10 num = 1177085018; clk = 0;  //0x4628E45A
  #10 clk = 1;
  #10 clk = 0;
  #10 clk = 1;
end
```

Figure 3 - Test-bench

If Sine function: Result = $r_{sa} * r_{cb} + r_{ca} * r_{sb}$

If Cosine function: Result = $r_{ca} * r_{cb} - r_{sa} * r_{sb}$

OUTPUT: Result of Sine-Cosine functions is fixed point 32-bit number under M format.

All above steps are also described as behind algorithm flowchart.

IV. CONCLUSIONS

In this paper, we present algorithm for computation of trigonometric functions such as Sine-Cosine functions. This algorithm is based on Table Lookup method and Polynomial Approximation method. We used Quartus II 13.0 and ModelSim-Altera 10.1d to verify our algorithm. Some results and test-bench are shown at Fig2 and Fig3. These results are checked with algorithm 64-bit on PC, the accuracy is fine with error is around $2^{-31} \approx 4.66 \times 10^{-10}$, this value also is the gap of M format representation.

TABLE IV. USING OF FIXED POINT UNITS

Fixed point unit	*Times of using*
Multiplier	6
Adder	3

TABLE V. MEMORY COST

Table	*Number of entries*	*Memory for each entries (in bytes)*	*Total table size (in bytes)*
Sine	201	4	804
Cosine	201	4	804
Total memory			1608 (< 2 Kbytes)

Table IV and V summarize computation and memory cost of the algorithm. We realize that the algorithm is not required many computations and only using fixed point units. Hence, it assures that computing speed is very fast. In fixed point computer, if fixed point multiplier is already available the algorithm will be developed easily; only one significant thing is creating tables. However, with constant resolution of M format $2^{-31} \approx 4.66 \times 10^{-10}$, we need to take care of the error in application which is required high accuracy.

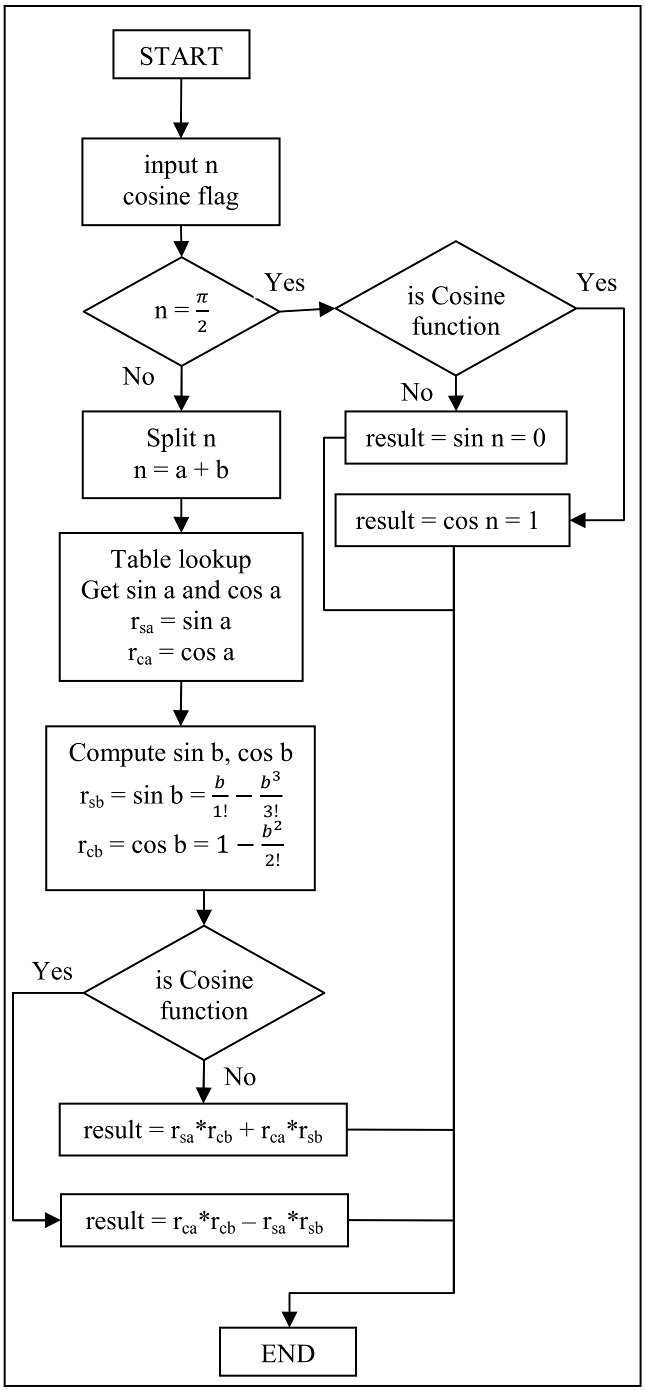


Figure 4 - Algorithm chart

REFERENCES

[1] P. T. P. Tang, “Table-lookup algorithms for elementary functions and their error analysis,” Argonne Nat. Lab. Rep., MCS-P194-1190, Jan.1991.

[2] P.T.P. Tang, ”Table-Lookup Algorithms for Elementary Functions and Their Error Analysis,” Proc. 10th Symp. Computer Arithmetic, pp. 232-236, 1991.

[3] S. Gal, “Computing elementary functions: A new approach for achieving high accuracy and good performance,” Lecture Notes in Computer Science 235, pp. 1-16, 1986.

[4] J. E. Volder, “The CORDIC trigonometric computing technique,” IRE Trans. Elec. Comput., vol. EC-9, pp. 227-231, Sept. 1960.

[5] S. Gal and B. Bachelis, “An accurate elementary mathematical library for the ieee floating point standard,” ACM Trans. Math. Software, vol.17, no. 1, pp. 26-45, Mar. 1991

[6] J. S. Walther, “A unified algorithm for elementary functions,” in Proc.Spring Joint Comput. Conf., pp. 379-385, 1971

[7] I. Koren and 0. Zinaty, “Evaluating elementary functions in a numerical coprocessor based on rational approximations,” IEEE Trans. Comput.,vol. 39, pp. 1030-1037, Aug. 1990.

[8] R. C. Agarwal, J. W. Cooley, F. G. Gustavson, J. B. Shearer, G. Slishman, and B. Tucker”, “New scalar and vector elementary functions for the IBM Systed/370,” IBM J. Res. Dev., pp. 126-144, Mar. 1986.

[9] P.W. Markstein, ”Computation of Elementary Functions on the IBM RISC System/6000 Processor,” IBM J. Research and Development, vol. 34, no. 1, pp. 111-119, Jan. 1990.

[10] W. Cody and W. Waite, Software Manual for the Elementay Functions. Englewood Cliffs, N.J.: Prentice Hall, 1980

[11] J.F. Hart et al., Computer Approximations. New York: John Wiley & Sons, 1968

[12] I. Koren, Computer Arithmetic Algorithms. Englewood Cliffs, N. J.: Prentice Hall, 1993.

[13] Xuan-Thuan Nguyen, Trong-Tu Bui, Huu-Thuan Huynh, Cong-Kha Pham, Duc-Hung Le, An Asic Implementation Of 16-Bit Fixed-Point Digital Signal Processor, International Conference on Advanced Computing and Applications (ACOMP), 2013.