Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных средств

Лабораторная работа

по курсу

«Параллельные и реконфигурируемые вычислительные системы»

|  |  |
| --- | --- |
| Выполнила: | магистрант группы 355841  А.В. Деркач |
| Проверил: | к.т.н., доцент  Н.А. Петровский |

Минск 2023

**1 ЗАДАНИЕ**

Подготовить сопроцессор CORDIC-алгоритма для вычисления математического выражения.

**2 ИСХОДНЫЕ ДАННЫЕ**

Математическое выражение (X, Y – входные значения, Z - результат):

Z = sin(X/Y)

**3 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

CORDIC (цифровой вычислитель поворота системы координат; метод «цифра за цифрой», алгоритм Волдера) – итерационный метод сведения прямых вычислений сложных функций к выполнению простых операций сложения и сдвига.

Идея метода заключается в сведении вычисления значений сложных (например, гиперболических) функций к набору простых шагов – сложению и сдвигу.

Такой подход особенно полезен при вычислении функций на устройствах с ограниченными вычислительными возможностями, такими как микроконтроллеры или программируемые логические матрицы (FPGA). Кроме того, поскольку шаги однотипны, то при аппаратной реализации алгоритм поддаётся развёртыванию в конвейер либо свертыванию в цикл.

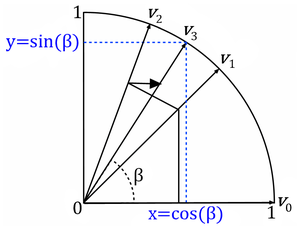


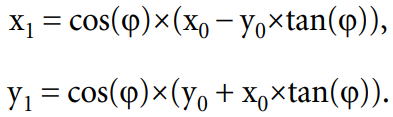
Рисунок 3.1 – Илюстрация алгоритма CORDIC

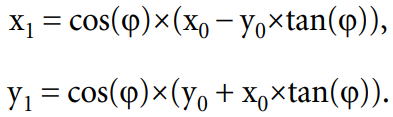
Рассмотрим суть этого алгоритма. Например, нам необходимо повернуть некий вектор с координатами (x0, y0) на угол φ, то есть нужно вычислить его новые координаты. Координаты x1 и y1 вычисляются по формулам:





После преобразования, эти формулы можно переписать в виде:





Если выбирать такой угол поворота, что tan(φ) = ±2–i, где i – целое число, то умножение значений x0 и y0 на tan(φ) превращается в простую операцию сдвига значений x0 и y0 на i разрядов (если представить их в двоичном счислении) вправо.

Если некий произвольный угол представить в виде суммы углов:



то операция поворота вектора будет состоять из последовательных элементарных поворотов. Также необходимо отметить, что направление поворота не влияет на множитель cos(φ), так как функция cos – четная. В формулах cos(φ) можно представить как cos(atan(2–i)). Так как i = 0, 1, 2…, то данная функция является сходящейся, результат обычно обозначается как Ki , равен ≈0,607 и называется коэффициентом деформации. Значит, помимо операций «сдвига» и «суммирование/вычитание» векторов, необходимо полученные координаты умножить на этот коэффициент деформации.

**4 ВЫПОЛНЕНИЕ РАБОТЫ**

Код реализации:

001. public class Cordic {

002.

003. private static final int CORDIC\_N = 16;

004. private static final int CORDIC\_K = 0x26DD3B6A;

005. private static final int[] CORDIC\_BETA = {

006. 0x3243F6A9, 0x1DAC6705, 0x0FADBAFD, 0x07F56EA7,

007. 0x03FEAB77, 0x01FFD55C, 0x00FFFAAB, 0x007FFF55,

008. 0x003FFFEB, 0x001FFFFD, 0x00100000, 0x00080000,

009. 0x00040000, 0x00020000, 0x00010000, 0x00008000

010. };

011.

012. private static double cordic(double phi, final boolean isSin) {

013.

014. phi %= 2 \* Math.PI;

015. if (phi < -Math.PI) phi += 2 \* Math.PI;

016. else if (phi > Math.PI) phi -= 2 \* Math.PI;

017. int x;

018.

019. int y = 0;

020. if (phi < -Math.PI / 2.0) {

021. phi += Math.PI;

022. x = -CORDIC\_K;

023. } else if (phi > Math.PI / 2.0) {

024. phi -= Math.PI;

025. x = -CORDIC\_K;

026. } else {

027. x = CORDIC\_K;

028. }

029.

030. int pp;

031. final int hi = (int) (Double.doubleToLongBits(phi) >> 52);

032. final int exp;

033. if ((exp = (hi & 0x7FF) - 1023) < -30) return isSin ? phi : 1.0;

034. if (exp > 0) return Double.NaN;

035. pp = (((int) (Double.doubleToLongBits(phi) >> 22) & 0x3FFFFFFF) | 0x40000000) >> -exp;

036. if (hi < 0) pp = -pp;

037.

038. for (int i = 0; i < CORDIC\_N; i++) {

039. if (pp >= 0) {

040. final int xx;

041. xx = x - (y >> i);

042. y += x >> i;

043. x = xx;

044. pp -= CORDIC\_BETA[i];

045. } else {

046. final int xx;

047. xx = x + (y >> i);

048. y -= x >> i;

049. x = xx;

050. pp += CORDIC\_BETA[i];

051. }

052. }

053. if (isSin) x = y;

054.

055. if (x == 0) return 0.0;

056.

057. final boolean neg;

058. if (neg = (x < 0)) x = -x;

059.

060. int xCopy = x;

061. x |= x >>> 1;

062. x |= x >>> 2;

063. x |= x >>> 4;

064. x |= x >>> 8;

065. x |= x >>> 16;

066. x = ~x;

067. x = ((x >> 1) & 0x55555555) + (x & 0x55555555);

068. x = ((x >> 2) & 0x33333333) + (x & 0x33333333);

069. x = ((x >> 4) & 0x0f0f0f0f) + (x & 0x0f0f0f0f);

070. x = ((x >> 8) & 0x00ff00ff) + (x & 0x00ff00ff);

071. final int shift = ((x >> 16) & 0x0000ffff) + (x & 0x0000ffff);

072. x = xCopy;

073.

074. final double res = Double.longBitsToDouble(((((long) x) << (shift + 33)) >>> 12) |

075. (((long) (1024 - shift)) << 52));

076. return neg ? -res : res;

077. }

078.

079. public static double sin(double a) {

080. return cordic(a, true);

081. }

082.

083. public static double cos(double a) {

084. return cordic(a, false);

085. }

086.

087. public static double abs(double a) {

088. return (a <= 0.0D) ? 0.0D - a : a;

089. }

090.

091. public static double divide(double x, double y) {

092. if (y == 0) {

093. return Double.POSITIVE\_INFINITY;

094. }

095.

096. double left = 0.0;

097. double right = Double.MAX\_VALUE;

098. double precision = 0.001;

099.

100. int sign = x \* y < 0 ? -1 : 1;

101. x = abs(x);

102. y = abs(y);

103.

104. while (true) {

105. double mid = left + ((right - left) \* 0.5);

106.

107. if (abs(y \* mid - x) <= precision) {

108. return mid \* sign;

109. }

110.

111. if (y \* mid < x) {

112. left = mid;

113. } else {

114. right = mid;

115. }

116. }

117. }

118.

119.

120. public static void main(String[] args) {

121. final double xInput = enterNumber("Введите значение X: ");

122. final double yInput = enterNumber("Введите значение Y: ");

123.

124. final double divisionCordicResult = divide(xInput, yInput);

125. final double divisionMathResult = xInput / yInput;

126. final double divisionError = divisionCordicResult - divisionMathResult;

127. System.out.printf("[BINARY] %12.10f/%12.10f равен %12.10f\n", xInput, yInput, divisionCordicResult);

128. System.out.printf("[MATH] %12.10f/%12.10f равен %12.10f\n", xInput, yInput, divisionMathResult);

129. System.out.printf("Погрешность деления равна %12.10f\n\n", divisionError);

130.

131. final double sinCordicResult = sin(xInput);

132. final double sinMathResult = Math.sin(xInput);

133. final double sinError = sinCordicResult - sinMathResult;

134.

135. System.out.printf("[CORDIC] sin(%12.10f) равен %12.10f\n", divisionCordicResult, sinCordicResult);

136. System.out.printf("[MATH] sin(%12.10f) равен %12.10f\n", divisionMathResult, sinMathResult);

137. System.out.printf("Погрешность sin равна %12.10f\n\n", sinError);

138. }

139.

140. public static double enterNumber(String message) {

141. Scanner scanner = new Scanner(System.in);

142. double number = 0.0;

143.

144. boolean validXInput = false;

145. while (!validXInput) {

146. try {

147. System.out.print(message);

148. number = scanner.nextDouble();

149. validXInput = true;

150. } catch (Exception e) {

151. System.out.println("Ошибка ввода. Введите корректное значение");

152. scanner.nextLine();

153. }

154. }

155.

156. return number;

157. }

158. }

Результаты выполнения:

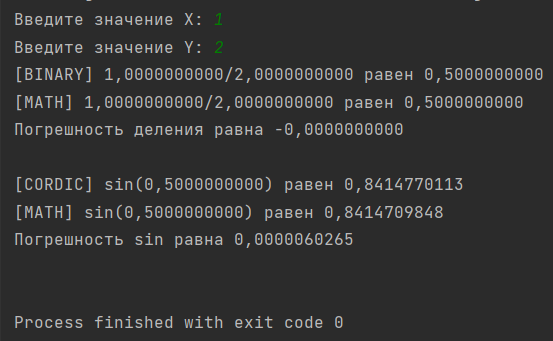


Рисунок 4.1 – Результат первого выполнения

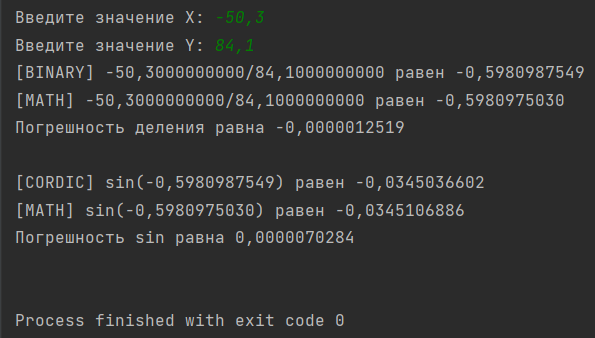


Рисунок 4.2 – Результат второго выполнения

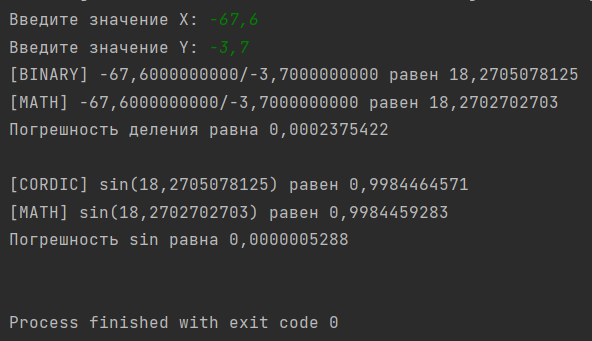


Рисунок 4.3 – Результат третьего выполнения

**5 ВЫВОДЫ**

В ходе выполнения лабораторной работы был подготовлен сопроцессор CORDIC-алгоритма для вычисления математического выражения Z = sin(X/Y) на языке программирования Java. Проведенные вычисления подтвердили эффективность CORDIC-алгоритма в вычислительных задачах, предоставляя точные результаты при сведении прямых вычислений сложных функций к выполнению простых операций сложения и сдвига. Полученные результаты подчеркивают применимость CORDIC-алгоритма на устройствах с ограниченными вычислительными возможностями.