*VLSI Design*, 2002 VOL. 15 (2), pp. 507–520

# A Synthesizable VHDL Model of the Exact Solution for Three-dimensional Hyperbolic Positioning System

RALPH BUCHER and D. MISRA*

*Department of Electrical and Computer Engineering, New Jersey Center for Wireless and Telecommunication, New Jersey Institute of Technology, Newark, NJ 07102, USA*

*(Received 1 August 2001; Revised 3 October 2001)*

This paper presents a synthesizable VHDL model of a three-dimensional hyperbolic positioning system algorithm. The algorithm obtains an exact solution for the three-dimensional location of a mobile given the locations of four fixed stations (like a global positioning system [GPS] satellite or a base station in a cell) and the signal time of arrival (TOA) from the mobile to each station. The detailed derivation of the steps required in the algorithm is presented. A VHDL model of the algorithm was implemented and simulated using the IEEE numeric_std package. Signals were described by a 32-bit vector. Simulation results predict location of the mobile is off by 1 m for best case and off by 36 m for worst case. A C++ program using real numbers was used as a benchmark for the accuracy and precision of the VHDL model. The model can be easily synthesized for low power hardware implementation.

*Keywords*: Wireless; Positioning system; Time of arrival (TOA); VHDL; GPS

## INTRODUCTION

Recently interests have emerged in using wireless position location for Intelligent Transportation System applications such as incident management, traffic routing, fleet management and E-911 telephone service [1]. Many designs have been proposed to solve the wireless position location problem [2–4]. Beacon location approach evaluates the signal strength from a mobile at many different known locations and determines the location of mobile. The other position locator approach is to evaluate the angle-of-arrival of a signal at two or more base stations, which determines the line of bearing and ultimately the mobile location is determined. The most widely used position location technique for geolocation of mobile users is the hyperbolic position location technique, also known as the time difference of arrival (TDOA) position location method. This technique utilizes cross-correlation process to calculate the difference in time of arrival (TOA) of a mobile signal at multiple (two or higher) pairs of stations. This delay defines a hyperbola of constant range difference from the receivers, which are located at the foci. Each TDOA measurement yields a hyperbolic curve along which the mobile may be positioned. When multiple stations are used, multiple hyperbolas are formed, and the intersection of the set of hyperbolas provides the estimated location of the source.

Many organizations are developing competing products to comply with the FCC's E-911 mandate, which requires US cellular carriers to provide location information of phone calls, effective October 2001. The accuracy required is 100 m or better. Many of these products will implement the above-mentioned TDOA technique for locating a mobile with varying degrees of accuracy. Methods for calculating the TDOA and mobile position have been reviewed previously [1,2]. Some methods calculate the two-dimensional position and others estimate the three-dimensional position depending on the degree of simplicity desired. In this paper, a more detailed derivation of a set of equations needed to locate the three-dimensional position of a mobile is presented. We have considered global positioning system (GPS) [5–8] to estimate the location. The nominal GPS operational constellation provides the user with between five and eight satellites visible from any point on the earth. For better accuracy four GPS satellite signals are typically used to compute positions in three dimensions. The detailed derivation in this work will be the basis for implementing a positioning algorithm in C++ and VHDL. The VHDL

*Corresponding author. E-mail: dmisra@njit.edu

ISSN 1065-514X print/ISSN 1563-5171 online © 2002 Taylor & Francis Ltd
DOI: 10.1080/1065514021000012129

version will utilize the IEEE numeric_std package so it can be synthesized into an ASIC by anyone seeking a hardware implementation.

## THE ALGORITHM

The essence of the TDOA technique is the equation for the distance between two points.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{1}$$

The distance between a mobile and a station is determined indirectly by measuring the time it takes for a signal to reach the station from the mobile. Multiplying the TOA $t$ by the signal velocity $c$ gives us the distance $d$. From now on, $R$ will be used to represent the distance $d$ since it is the more commonly used notation in TDOA literature.

We need to solve for the three unknowns $x$, $y$ and $z$ (mobile position). Therefore, Eq. (1) is expanded to three equations when the specific locations of three satellites $i$, $j$ and $k$ are given. This requirement can be easily met since GPS satellites broadcast their exact locations.

$$ct_i = R_i = \sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} \tag{2}$$

$$ct_j = R_j = \sqrt{(x_j - x)^2 + (y_j - y)^2 + (z_j - z)^2} \tag{3}$$

$$ct_k = R_k = \sqrt{(x_k - x)^2 + (y_k - y)^2 + (z_k - z)^2} \tag{4}$$

where, $x_i$, $y_i$, $z_i$, $x_j$, $y_j$, $z_j$ and $x_k$, $y_k$, $z_k$ are the position of $i$th, $j$th and $k$th satellite, respectively and these positions vary with time.

Unfortunately, solving the three equations for three unknowns will not lead to a simple and satisfactory solution because of the square root terms. The solution can be simplified by adding another satellite $l$ for an additional equation. In addition, the accuracy of the mobile position will be further improved if four equations are used. This requirement is easily met since four GPS satellites are guaranteed to be in the horizon of any location on earth [9]. The four equations will be combined to form expressions for time difference of arrivals (TDOAs) $R_{ij}$, $R_{ik}$, $R_{kj}$ and $R_{kl}$.

$$R_i - R_j = R_{ij} = \sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} - \sqrt{(x_j - x)^2 + (y_j - y)^2 + (z_j - z)^2} \tag{5}$$

$$R_i - R_k = R_{ik} = \sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} - \sqrt{(x_k - x)^2 + (y_k - y)^2 + (z_k - z)^2} \tag{6}$$

$$R_k - R_j = R_{kj} = \sqrt{(x_k - x)^2 + (y_k - y)^2 + (z_k - z)^2} - \sqrt{(x_j - x)^2 + (y_j - y)^2 + (z_j - z)^2} \tag{7}$$

$$R_k - R_l = R_{kl} = \sqrt{(x_k - x)^2 + (y_k - y)^2 + (z_k - z)^2} - \sqrt{(x_l - x)^2 + (y_l - y)^2 + (z_l - z)^2} \tag{8}$$

Moving one square root term to the other side gives us:

$$R_{ij} - \sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} = -\sqrt{(x_j - x)^2 + (y_j - y)^2 + (z_j - z)^2} \tag{9}$$

$$R_{ik} - \sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} = -\sqrt{(x_k - x)^2 + (y_k - y)^2 + (z_k - z)^2} \tag{10}$$

$$R_{kj} - \sqrt{(x_k - x)^2 + (y_k - y)^2 + (z_k - z)^2} = -\sqrt{(x_j - x)^2 + (y_j - y)^2 + (z_j - z)^2} \tag{11}$$

$$R_{kl} - \sqrt{(x_k - x)^2 + (y_k - y)^2 + (z_k - z)^2} = -\sqrt{(x_l - x)^2 + (y_l - y)^2 + (z_l - z)^2} \tag{12}$$

Squaring both sides produces the following set of equations:

$$R_{ij}^2 - 2R_{ij}\sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} + (x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2 = (x_j - x)^2 + (y_j - y)^2 + (z_j - z)^2 \tag{13}$$

$$R_{ik}^2 - 2R_{ik}\sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} + (x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2 = (x_k - x)^2 + (y_k - y)^2 + (z_k - z)^2 \tag{14}$$

$$R_{kj}^2 - 2R_{kj}\sqrt{(x_k - x)^2 + (y_k - y)^2 + (z_k - z)^2} + (x_k - x)^2 + (y_k - y)^2 + (z_k - z)^2 = (x_j - x)^2 + (y_j - y)^2 + (z_j - z)^2 \tag{15}$$

$$R_{kl}^2 - 2R_{kl}\sqrt{(x_k - x)^2 + (y_k - y)^2 + (z_k - z)^2} + (x_k - x)^2 + (y_k - y)^2 + (z_k - z)^2 = (x_l - x)^2 + (y_l - y)^2 + (z_l - z)^2 \tag{16}$$

Expanding the squared terms to the left of the square root term produces:

$$R_{ij}^2 - 2R_{ij}\sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} + x_i^2 - 2x_ix + x^2 + y_i^2 - 2y_iy + y^2 + z_i^2 - 2z_iz + z^2 = x_j^2 - 2x_jx + x^2 + y_j^2 - 2y_jy + y^2 + z_j^2 - 2z_jz + z^2 \tag{17}$$

$$R_{ik}^2 - 2R_{ik}\sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} + x_i^2 - 2x_ix + x^2 + y_i^2 - 2y_iy + y^2 + z_i^2 - 2z_iz + z^2 = x_k^2 - 2x_kx + x^2 + y_k^2 - 2y_ky + y^2 + z_k^2 - 2z_kz + z^2 \tag{18}$$

$$R_{kj}^2 - 2R_{kj}\sqrt{(x_k - x)^2 + (y_k - y)^2 + (z_k - z)^2} + x_k^2 - 2x_kx + x^2 + y_k^2 - 2y_ky + y^2 + z_k^2 - 2z_kz + z^2 = x_j^2 - 2x_jx + x^2 + y_j^2 - 2y_jy + y^2 + z_j^2 - 2z_jz + z^2 \tag{19}$$

$$R_{kl}^2 - 2R_{kl}\sqrt{(x_k - x)^2 + (y_k - y)^2 + (z_k - z)^2} + x_k^2 - 2x_kx + x^2 + y_k^2 - 2y_ky + y^2 + z_k^2 - 2z_kz + z^2 = x_l^2 - 2x_lx + x^2 + y_l^2 - 2y_ly + y^2 + z_l^2 - 2z_lz + z^2 \tag{20}$$

Eliminating the $x^2$, $y^2$ and $z^2$ terms reduces the equation set to:

$$R_{ij}^2 - 2R_{ij}\sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} + x_i^2 - 2x_ix + y_i^2 - 2y_iy + z_i^2 - 2z_iz = x_j^2 - 2x_jx + y_j^2 - 2y_jy + z_j^2 - 2z_jz \tag{21}$$

$$R_{ik}^2 - 2R_{ik}\sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} + x_i^2 - 2x_ix + y_i^2 - 2y_iy + z_i^2 - 2z_iz = x_k^2 - 2x_kx + y_k^2 - 2y_ky + z_k^2 - 2z_kz \tag{22}$$

$$R_{kj}^2 - 2R_{kj}\sqrt{(x_k - x)^2 + (y_k - y)^2 + (z_k - z)^2} + x_k^2 - 2x_kx + y_k^2 - 2y_ky + z_k^2 - 2z_kz = x_j^2 - 2x_jx + y_j^2 - 2y_jy + z_j^2 - 2z_jz \tag{23}$$

$$R_{kl}^2 - 2R_{kl}\sqrt{(x_k - x)^2 + (y_k - y)^2 + (z_k - z)^2} + x_k^2 - 2x_kx + y_k^2 - 2y_ky + z_k^2 - 2z_kz = x_l^2 - 2x_lx + y_l^2 - 2y_ly + z_l^2 - 2z_lz \tag{24}$$

Shifting all but the square root term to the right and combining similar terms produces:

$$\sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} = [R_{ij}^2 + x_i^2 - x_j^2 + y_i^2 - y_j^2 + z_i^2 - z_j^2 + 2x_jx - 2x_ix + 2y_jy - 2y_iy + 2z_jz - 2z_iz]/2R_{ij} \tag{25}$$

$$\sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} = [R_{ik}^2 + x_i^2 - x_k^2 + y_i^2 - y_k^2 + z_i^2 - z_k^2 + 2x_kx - 2x_ix + 2y_ky - 2y_iy + 2z_kz - 2z_iz]/2R_{ik} \tag{26}$$

$$\sqrt{(x_k - x)^2 + (y_k - y)^2 + (z_k - z)^2} = [R_{kj}^2 + x_k^2 - x_j^2 + y_k^2 - y_j^2 + z_k^2 - z_j^2 + 2x_jx - 2x_kx + 2y_jy - 2y_ky + 2z_jz - 2z_kz]/2R_{kj} \tag{27}$$

$$\sqrt{(x_k - x)^2 + (y_k - y)^2 + (z_k - z)^2} = [R_{kl}^2 + x_k^2 - x_l^2 + y_k^2 - y_l^2 + z_k^2 - z_l^2 + 2x_lx - 2x_kx + 2y_ly - 2y_ky + 2z_lz - 2z_kz]/2R_{kl} \tag{28}$$

The equation set can now be simplified by substituting $x_{ji}$ for $x_j - x_i$, $y_{ji}$ for $y_j - y_i$ and so on.

$$\sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} = [R_{ij}^2 + x_i^2 - x_j^2 + y_i^2 - y_j^2 + z_i^2 - z_j^2 + 2x_{ji}x + 2y_{ji}y + 2z_{ji}z]/2R_{ij} \tag{29}$$

$$\sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} = [R_{ik}^2 + x_i^2 - x_k^2 + y_i^2 - y_k^2 + z_i^2 - z_k^2 + 2x_{ki}x + 2y_{ki}y + 2z_{ki}z]/2R_{ik} \tag{30}$$

$$\sqrt{(x_k - x)^2 + (y_k - y)^2 + (z_k - z)^2} = [R_{kj}^2 + x_k^2 - x_j^2 + y_k^2 - y_j^2 + z_k^2 - z_j^2 + 2x_{jk}x + 2y_{jk}y + 2z_{jk}z]/2R_{kj} \tag{31}$$

$$\sqrt{(x_k - x)^2 + (y_k - y)^2 + (z_k - z)^2} = [R_{kl}^2 + x_k^2 - x_l^2 + y_k^2 - y_l^2 + z_k^2 - z_l^2 + 2x_{lk}x + 2y_{lk}y + 2z_{lk}z]/2R_{kl} \tag{32}$$

Equations (5)–(8) are now in a useful arrangement. Equations (29)–(32), when squared, are intersecting hyperboloids. By equating Eqs. (29) and (30) to form Eq. (33), we can derive a plane equation in the form of $y = Ax + By + C$ by rearranging the terms as shown in Eqs. (34) and (35).

$$[R_{ij}^2 + x_i^2 - x_j^2 + y_i^2 - y_j^2 + z_i^2 - z_j^2 + 2x_{ji}x + 2y_{ji}y + 2z_{ji}z]/2R_{ij} = [R_{ik}^2 + x_i^2 - x_k^2 + y_i^2 - y_k^2 + z_i^2 - z_k^2 + 2x_{ki}x + 2y_{ki}y + 2z_{ki}z]/2R_{ik} \tag{33}$$

$$R_{ik}[R_{ij}^2 + x_i^2 - x_j^2 + y_i^2 - y_j^2 + z_i^2 - z_j^2]/2 - R_{ij}[R_{ik}^2 + x_i^2 - x_k^2 + y_i^2 - y_k^2 + z_i^2 - z_k^2]/2 = R_{ij}[x_{ki}x + y_{ki}y + z_{ki}z] - R_{ik}[x_{ji}x + y_{ji}y + z_{ji}z] \tag{34}$$

$$x[R_{ij}x_{ki} - R_{ik}x_{ji}] + y[R_{ij}y_{ki} - R_{ik}x_{ji}] + z[R_{ij}z_{ki} - R_{ik}z_{ji}] = R_{ik}[R_{ij}^2 + x_i^2 - x_j^2 + y_i^2 - y_j^2 + z_i^2 - z_j^2]/2 - R_{ij}[R_{ik}^2 + x_i^2 - x_k^2 + y_i^2 - y_k^2 + z_i^2 - z_k^2]/2 \tag{35}$$

Equation (35) is now in the desired form of a plane equation as follows:

$$y = Ax + Bz + C \tag{36}$$

where

$$A = \left[\frac{R_{ik}x_{ji} - R_{ij}x_{ki}}{R_{ij}y_{ki} - R_{ik}y_{ji}}\right] \tag{37}$$

and

$$B = \left[\frac{R_{ik}z_{ji} - R_{ij}z_{ki}}{R_{ij}y_{ki} - R_{ik}y_{ji}}\right] \tag{38}$$

and

$$C = \frac{R_{ik}[R_{ij}^2 + x_i^2 - x_j^2 + y_i^2 - y_j^2 + z_i^2 - z_j^2] - R_{ij}[R_{ik}^2 + x_i^2 - x_k^2 + y_i^2 - y_k^2 + z_i^2 - z_k^2]}{2[R_{ij}y_{ki} - R_{ik}y_{ji}]} \tag{39}$$

Similarly, equating Eqs. (31) and (32) produces a second plane equation $y = Dx + Ey + F$. The resulting set of equations are:

$$y = Dx + Ez + F \tag{40}$$

where

$$D = \left[\frac{R_{kl}x_{jk} - R_{kj}x_{lk}}{R_{kj}y_{lk} - R_{kl}y_{jk}}\right] \tag{41}$$

and

$$E = \left[\frac{R_{kl}z_{jk} - R_{kj}z_{lk}}{R_{kj}y_{lk} - R_{kl}y_{jk}}\right] \tag{42}$$

and

$$F = \frac{R_{kl}[R_{kj}^2 + x_k^2 - x_j^2 + y_k^2 - y_j^2 + z_k^2 - z_j^2] - R_{kj}[R_{kl}^2 + x_k^2 - x_l^2 + y_k^2 - y_l^2 + z_k^2 - z_l^2]}{2[R_{kj}y_{lk} - R_{kl}y_{jk}]} \tag{43}$$

Equating the plane Eqs. (36) and (40) produces a linear equation for $x$ in terms of $z$.

$$Ax + Bx + C = Dx + Ez + F \tag{44}$$

$$x = Gz + H \tag{45}$$

where

$$G = \frac{E - B}{A - D} \tag{46}$$

and

$$H = \frac{F - C}{A - D} \tag{47}$$

Substituting Eq. (45) back into Eq. (36) produces a linear equation for $y$ in terms of $z$.

$$y = A(Gz + H) + Bz + C \tag{48}$$

$$y = Iz + J \tag{49}$$

where

$$I = AG + B \tag{50}$$

and

$$J = AH + C \tag{51}$$

Equations (45) and (49) are now substituted back into Eq. (30) to derive the position $z$.

$$2R_{ik}\sqrt{(x_i-(Gz+H))^2+(y_i-(Iz+J))^2+(z_i-z)^2}$$
$$=[R_{ik}^2+x_i^2-x_k^2+y_i^2-y_k^2+z_i^2-z_k^2$$
$$+2x_{ki}(Gz+H)+2y_{ki}(Iz+J)+2z_{ki}z] \tag{52}$$

$$2R_{ik}\sqrt{(G^2z^2-2Gz(x_i-H)+(x_i-H)^2)+(I^2z^2-2Iz(y_i-J)+(y_i-J)^2)+(z^2-2z_iz+z_i^2)}$$
$$=Lz+K \tag{53}$$

where

$$K=R_{ik}^2+x_i^2-x_k^2+y_i^2-y_k^2+z_i^2-z_k^2+2x_{ki}H$$
$$+2y_{ki}J \tag{54}$$

and

$$L=2[x_{ki}G+y_{ki}I+2z_{ki}] \tag{55}$$

$$4R_{ik}^2[G^2z^2+I^2z^2+z^2-2Gz(x_i-H)-2Iz(y_i-J)$$
$$-2z_iz+(x_i-H)^2+(y_i-J)^2+z_i^2]$$
$$=L^2z^2+2KLz+K^2 \tag{56}$$

$$4R_{ik}^2[G^2+I^2+1]z^2-8R_{ik}^2[G(x_i-H)+I(y_i-J)$$
$$+z_i]z+4R_{ik}^2[(x_i-H)^2+(y_i-J)^2+z_i^2]$$
$$=L^2z^2+2KLz+K^2 \tag{57}$$

To obtain $z$, Eq. (57) is rearranged into a binomial equation.

$$Mz^2-Nz+O=0 \tag{58}$$

where

$$M=4R_{ik}^2[G^2+I^2+1]-L^2 \tag{59}$$

and

$$N=8R_{ik}^2[G(x_i-H)+I(y_i-J)+z_i]+2LK \tag{60}$$

and

$$O=4R_{ik}^2[(x_i-H)^2+(y_i-J)^2+z_i^2]-K^2 \tag{61}$$

The solution for $z$ is:

$$z=\frac{N}{2M}\pm\sqrt{\left(\frac{N}{2M}\right)^2-\frac{O}{M}} \tag{62}$$

The $z$ coordinate can be put back into the linear Eqs. (45) and (49) to solve for the coordinates $x$ and $y$.

## VHDL MODEL

The equations for the $x$, $y$ and $z$ position of the mobile was modeled using VHDL. The numeric_std package was used to construct the VHDL model that was readily synthesized into a low power digital circuit. The details of the circuit are beyond the scope of this paper. The input signals of the model are the $x$, $y$, $z$ positions of four GPS satellites $i$, $j$, $k$, $l$ in meters, and the signal TOAs from the individual satellites to the mobile in nanoseconds. The input signal assignments are *xi*, *yi*, *zi*, *ti*, *xj*, *yj*, *zj*, *tj*, *xk*, *yk*, *zk*, *tk*, *xl*, *yl*, *zl* and *tl*.

GPS satellite altitudes are approximately 10,900 nautical miles (20,186,800 m). Therefore, the TOA range is roughly 6,700,000–7,600,000 ns. This means the input signals can be adequately described by a 32-bit vector. In order to perform signed arithmetic operations, the input signal assignments are of type SIGNED. The binary representation for negative numbers is 2's complement. The TDOAs are converted to distances by multiplying them by the binary representation of 100,000, and then dividing the result by the binary representation of 333,564 ns/m.

Since all signal and variable assignments are vectors representing integers, a method for maintaining adequate precision in divide and square root operations is needed. This will be achieved by multiplying the numerator by the binary representation of $1.0\times10^{10}$ in divide operations. This method is preferred to using decimal point notation to decrease the complexity of the model. However, the length of the vectors increases for successive multiplication operations, leading to a 200-bit vector for the interim value $O$.

The numeric_std package does not contain an overloaded square root operator. Therefore, Dijkstra's bisection algorithm [10] is used to compute the integer square root of a positive integer represented by a 64-bit vector. Sixty four bits is deemed adequate since the position $z$ and the square root term cannot be larger than 32 bits by definition.

The square root operation gives two values for $z$, so the output signals $z1$, $z2$, $x1$, $x2$, $y1$, $y2$ are for two possible mobile positions. The $z$ value representing the mobile position can be determined by using a fifth satellite, or checking if the value is in the horizon of the four satellites relative to earth. The details of the VHDL model of the algorithm are provided in Appendix A.

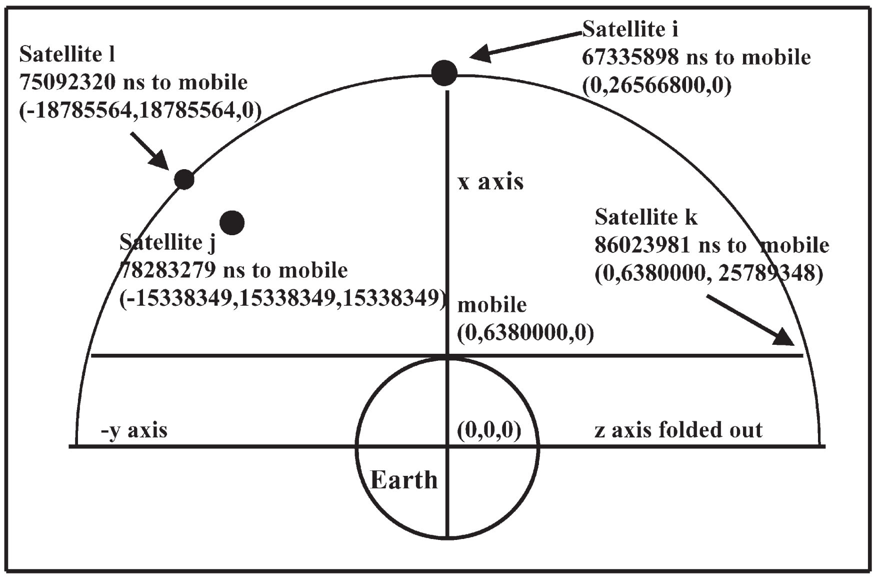

FIGURE 1 A real life situation with the satellite positions and TOA from the satellite at the mobile is specified in nanoseconds.

## TESTING THE MODEL

Accolade's demonstration edition of PeakVHDL was used to compile the model and run simulations. The model was also compiled and synthesized with Mentor Graphics' AUTOLOGIC II [11] in low power mode to ensure the model was synthesizable. A high level schematic was generated (not shown here) using AUTOLOGIC's default component library.

Figures 1 and 2 show two different real life situations. The satellite positions and TOA from the satellite at the mobile in nanoseconds are specified in the figures. The known position of a mobile was used to determine the input test data for *ti*, *tj*, *tk* and *tl*. The test data for the satellite positions *xi*, *yi*, *zi*, *xj*, *yj*, *zj*, *xk*, *yk*, *zk*, *xl*, *yl* and *zl* are realistic numbers.

The VHDL test benches representing the real life situations in Figs. 1 and 2 were used to validate the model. The test bench converted the base 10 numbers in Fig. 1 or Fig. 2 to binary numbers and outputted the *x*, *y* and *z* positions of the mobile in binary and base 10 format. Two test benches (Appendix B and C) representing two sets of

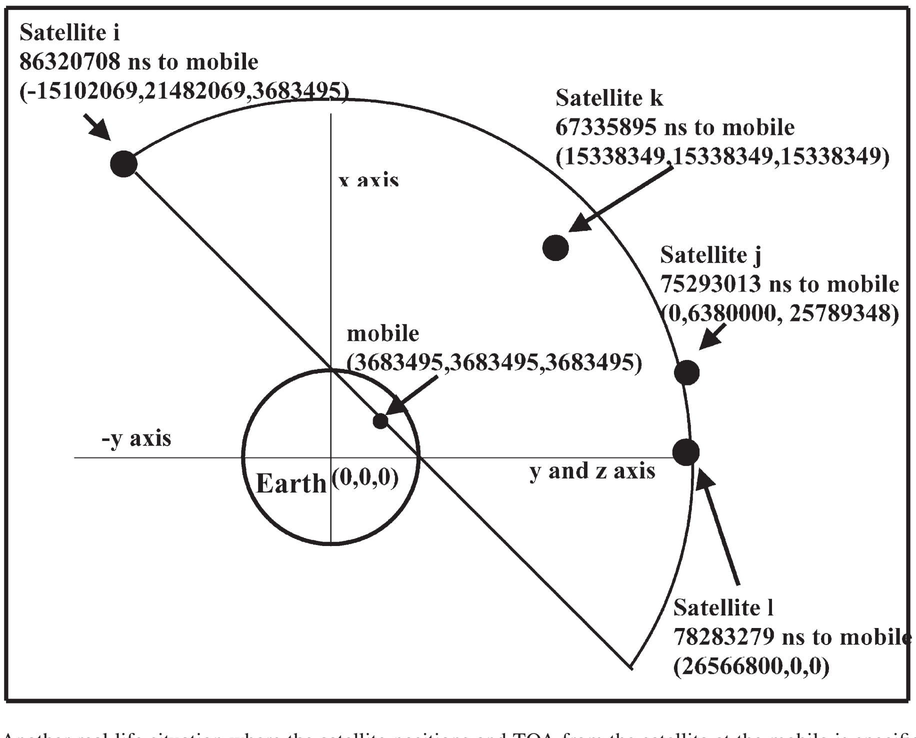

FIGURE 2 Another real life situation where the satellite positions and TOA from the satellite at the mobile is specified in nanoseconds.

TABLE I The results of VHDL simulation after applying the first test bench (Appendix B) for Fig. 1

| Signal | Value |
|---|---|
| int_$x$2 | − 1 |
| int_$y$2 | 6379964 |
| int_$z$2 | 0 |
| $x$2 | 11111111111111111111111111111111 |
| $y$2 | 00000000011000010101100110111100 |
| $z$2 | 00000000000000000000000000000000 |

data were used to simulate the model. Two C++ programs (Appendix D and E) using real numbers were used as a benchmark for the accuracy and precision of the VHDL model. The results of VHDL simulation displayed in Table I after applying the first test bench (Appendix B) for Fig. 1. Table II shows the results of VHDL simulation after applying the second test bench (Appendix C) for Fig. 2. Results from the C++ program (Appendix D) which corresponds to the first test bench are shown in Table III whereas Table IV shows the results from the C++ program (Appendix E) which corresponds to the second test bench.

The C++ program and VHDL model produced the same results. This means the VHDL model can produce the coordinates as accurate as a GPS utilizing a general purpose microprocessor with a 32-bit IEEE floating point ALU. For Fig. 1, the $y$ position was off by 36 m, and the $x$ position was off by 1 m. For Fig. 2, the $x$ position was off by 1 m. The VHDL model's accuracy could be improved by extending the precision beyond the ten decimal points as it is currently constructed. However, the number of gates will increase in the synthesized circuit.

## SUMMARY

In summary, we have presented a synthesizable VHDL model of a three-dimensional hyperbolic positioning system algorithm. We obtained the exact solution for the three-dimensional location of a mobile given the locations of four fixed GPS satellites or a base station in a cell and the signal TOA from the mobile to each station. The algorithm was implemented using a VHDL model of and simulated using the IEEE numeric_std package. Simulation results for two different situations predict location of the mobile is off by 1 m for best case and off by 36 m for worst case. A C++ program using real numbers was used as a benchmark for the accuracy and precision of the VHDL model. The model can be easily synthesized for low power hardware implementation.

TABLE II The results of VHDL simulation after applying the second test bench (Appendix C) for Fig. 2

| Signal | Value |
|---|---|
| int_$x$1 | 3683494 |
| int_$y$1 | 3683495 |
| int_$z$1 | 3682495 |
| $x$1 | 00000000001110000011010010100110 |
| $y$1 | 00000000001110000011010010100111 |
| $z$1 | 00000000001110000011010010100111 |

TABLE III The results of C++ program (Appendix D) which corresponds to the first test bench (Fig. 1)

| Signal | Value |
|---|---|
| $x$1 | − 1 |
| $y$1 | 6379964 |
| $z$1 | 0 |

TABLE IV The results of C++ program (Appendix E) which corresponds to the second test bench (Fig. 2)

| Signal | Value |
|---|---|
| $x$1 | 3683494 |
| $y$1 | 3683495 |
| $z$1 | 3683495 |

### Acknowledgements

Authors will acknowledge the financial support from the New Jersey Center for Wireless and Telecommunications (NJCWT) for this work.

### References

[1] Fang, B.T. (1990) "Simple solutions for a hyperbolic and related position fixes", *IEEE Trans. Aerosp. Elect. Systems* **26**(5), 748–753.

[2] Krizman, K.J., Biedka, T.E. and Rappaport, T.S. (1997) "Wireless position location: Fundamentals, implementation strategies, and sources of error, invited paper", Proceedings of Virginia Tech's Seventh Symposium on Wireless Personal Communications, Blacksburg, VA, June 11–13.

[3] Rappaport, T.S., Reed, J.H. and Woerner, B.D. (1996) "Position location using Wireless Communications on Highways of the Future, invited paper", *IEEE Commun. Mag.* **34**(10), 33–41.

[4] Liberti, J.C. and Rappaport, T.S. (1999), Chapter 10, Smart Antennas for Wireless Communications: IS-95 and Third Generation CDMA Applications (Prentice Hall, Englewood Cliffs, NJ).

[5] Yacoub, M.D. (1993) Foundations of Mobile Radio Engineering (CRC Press, Boca Raton, FL).

[6] Klepczynski, W.J. (1996) "GPS for Precise Time and Time Interval Measurement, Global Positioning System: Theory and Applications", In: Parkinson, B.W. and Spilker, Jr., J.J., eds, Vol. 164 of Progress in Astronautics and Aeronautics (American Institute of Aeronautics and Astronautics, Washington, DC) **Vol. II**, p 483.

[7] Stilp, L.A. (1995) "Time Difference of Arrival Technology for Locating Narrowband Cellular Signals." Proceedings of the SPIE Conference on Wireless Technologies and Services for Cellular and Personal Communication Services held in Philadelphia, PA, October 2226, *SPIE Proceedings Vol. 2602*, 134 pp.

[8] Rao, B.D. and Hari, K.V. (1989) "Performance Analysis of Root-Music", *IEEE Trans. Acoustics, Speech, Signal Processing* **37**(12), 1939.

[9] Leick, A. (1995) GPS Satellite Surveying, 2nd Ed. (Wiley, New York).

[10] Zachary, J.L. (1996) Introduction to Scientific Programming Computational Problem Solving Using: Maple and C (Springer-Verlag, New York).

[11] Autologic-II, User Manual, Mentor Graphics Corp, Wilsonville, OR USA.

## APPENDIX A

```
--VHDL model for computing x, y, z position of mobile given four satellite

--positions and TOAs from satellites to mobile

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity hyperbolic is
port(xi,xj,xk,xl,yi,yj,yk,yl,zi,zj,zk,zl,ti,tk,tj,tl: in SIGNED(31 downto 0);
     x1,x2,y1,y2,z1,z2: out SIGNED(31 downto 0));
end hyperbolic;

architecture behave of hyperbolic is

signal o: SIGNED(199 downto 0);
signal n: SIGNED(195 downto 0);
signal m: SIGNED(191 downto 0);
signal c,f,l: SIGNED(95 downto 0);
signal s12,s16,s21,s22,s24: SIGNED(131 downto 0);
signal s9,s11,s13,s15,s23,k: SIGNED(99 downto 0);
signal s1,s2,s3,s4,s5,s6,s7,s8,s10,s14,s17,s18,s19,s20,s26,s27,s29,s30,
       a,b,d,e,g,h,i,j,yi2,yj2,yk2,yl2,xi2,xj2,xk2,xl2,zi2,zj2,zk2,zl2,
       rij2,rik2,rkl2,rkj2: SIGNED(63 downto 0);
signal rij,rik,rkj,rkl,xji,xki,xjk,xlk,yji,yki,yjk,ylk,zji,zki,zjk,zlk,
       light,thou,root,s28,s31,s32,s33,s34,s35,s36: SIGNED(31 downto 0);
signal one_e_10: SIGNED(35 downto 0):="001001010100000010111110010000000000";

begin

light<=to_signed(333564,32);            thou<=to_signed(100000,32);

s1<=abs(thou*(ti-tj));  s5<=s1/light;    rij<=resize(s5,32);     rij2<=rij*rij;
s2<=abs(thou*(ti-tk));  s6<=s2/light;    rik<=resize(s6,32);     rik2<=rik*rik;
s3<=abs(thou*(tk-tl));  s7<=s3/light;    rkl<=resize(s7,32);     rkl2<=rkl*rkl;
s4<=abs(thou*(tk-tj));  s8<=s4/light;    rkj<=resize(s8,32);     rkj2<=rkj*rkj;

xji<=xj-xi; yji<=yj-yi; zji<=zj-zi;      yi2<=yi*yi; xi2<=xi*xi; zi2<=zi*zi;
xki<=xk-xi; yki<=yk-yi; zki<=zk-zi;      yj2<=yj*yj; xj2<=xj*xj; zj2<=zj*zj;
xlk<=xl-xk; ylk<=yl-yk; zlk<=zl-zk;      yk2<=yk*yk; xk2<=xk*xk; zk2<=zk*zk;
xjk<=xj-xk; yjk<=yj-yk; zjk<=zj-zk;      yl2<=yl*yl; xl2<=xl*xl; zl2<=zl*zl;

s9 <=one_e_10*(rik*xji-rij*xki);         s10<=rij*yki-rik*yji;
s11<=one_e_10*(rik*zji-rij*zki);         s13<=one_e_10*(rkl*xjk-rkj*xlk);
s14<=rkj*ylk-rkl*yjk;                    s15<=one_e_10*(rkl*zjk-rkj*zlk);

s17<=rij2+xi2-xj2+yi2-yj2+zi2-zj2;       s18<=rik2+xi2-xk2+yi2-yk2+zi2-zk2;
s19<=rkj2+xk2-xj2+yk2-yj2+zk2-zj2;       s20<=rkl2+xk2-xl2+yk2-yl2+zk2-zl2;
s12<=one_e_10*(rik*s17-rij*s18);         s16<=one_e_10*(rkl*s19-rkj*s20);
s21<=SHIFT_RIGHT(s12,1);                 s22<=SHIFT_RIGHT(s16,1);

a<=resize(s9/s10,64);   b<=resize(s11/s10,64);  c<=resize(s21/s10,96);
d<=resize(s13/s14,64);  e<=resize(s15/s14,64);  f<=resize(s22/s14,96);
```

```
s23<=one_e_10*(e-b);                s24<=one_e_10*(f-c);

g<=resize(s23/(a-d),64);            h<=resize(s24/(a-d),64);
i<=resize(((a*g)/one_e_10)+b,64);   j<=resize(((a*h)/one_e_10)+c,64);

k<=s18*one_e_10+SHIFT_LEFT(j*yki,1)+SHIFT_LEFT(h*xki,1);

l<=SHIFT_LEFT(g*xki+i*yki+zki*one_e_10,1);

m<=SHIFT_LEFT(rik2*(g*g+i*i+one_e_10*one_e_10),2)-l*l;

s26<=resize(one_e_10*xi-h,64);      s27<=resize(one_e_10*yi-j,64);

n<=SHIFT_LEFT(rik2*(g*s26+i*s27+zi*one_e_10*one_e_10),3)+SHIFT_LEFT(l*k,1);

o<=SHIFT_LEFT(rik2*(s26*s26+s27*s27+zi*zi*one_e_10*one_e_10),2)-k*k;

s28<=resize(SHIFT_RIGHT(n/m,1),32);
s29<=resize(o/m,64);
s30<=s28*s28-S29;

squareroot:process (s30)
      variable q,r,s,t:signed(63 downto 0);
      begin
      q:=s30;
      s:=to_signed(0,64);
      r:=to_signed(0,64);
      t:="0100000000000000000000000000000000000000000000000000000000000000";
      for j in 1 to 32 loop
      s:=r+t;
      if s<=q then
            q:=q-s;
            r:=s+t;
      end if;
      r:=shift_right(r,1);
      t:=shift_right(t,2);
      end loop;
      root<=resize(r,32);
end process squareroot;

s31<=S28+root;                                  z1<=s31;
s32<=resize((g*s31+h)/one_e_10,32);             x1<=s32;
s33<=resize((a*s32+b*s31+c)/one_e_10,32);       y1<=s33;
s34<=s28-root;                                  z2<=s34;
s35<=resize((g*s34+h)/one_e_10,32);             x2<=s35;
s36<=resize((a*s35+b*s34+c)/one_e_10,32);       y2<=S36;

end behave;
```

## APPENDIX B

```
--VHDL test program for hyperbolic.vhd
--TEST CASE 1

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity test is
end test;

architecture stimulus of test is

component hyperbolic
port(xi,xj,xk,xl,yi,yj,yk,yl,zi,zj,zk,zl,ti,tk,tj,tl:in SIGNED(31 downto 0);
     x1,x2,y1,y2,z1,z2: out SIGNED(31 downto 0));
end component;

signal ti:SIGNED(31 downto 0):=to_signed(67335898,32);
signal tj:SIGNED(31 downto 0):=to_signed(78283279,32);
signal tk:SIGNED(31 downto 0):=to_signed(86023981,32);
signal tl:SIGNED(31 downto 0):=to_signed(75092320,32);

signal xi:SIGNED(31 downto 0):=to_signed(0,32);
signal xj:SIGNED(31 downto 0):="11111111000101011111010010010011";-- (-15338349)
signal xk:SIGNED(31 downto 0):=to_signed(0,32);
signal xl:SIGNED(31 downto 0):="11111110111000010101101011100100";-- (-18785564)

signal yi:SIGNED(31 downto 0):=to_signed(26566800,32);
signal yj:SIGNED(31 downto 0):=to_signed(15338349,32);
signal yk:SIGNED(31 downto 0):=to_signed(6380000,32);
signal yl:SIGNED(31 downto 0):=to_signed(18785564,32);

signal zi:SIGNED(31 downto 0):=to_signed(0,32);
signal zj:SIGNED(31 downto 0):=to_signed(15338349,32);
signal zk:SIGNED(31 downto 0):=to_signed(25789348,32);
signal zl:SIGNED(31 downto 0):=to_signed(0,32);

signal x1,x2,y1,y2,z1,z2:SIGNED(31 downto 0):=to_signed(0,32);

signal int_x1,int_x2,int_y1,int_y2,int_z1,int_z2:integer:=0;

begin

s1: hyperbolic port map (xi => xi, xj => xj, xk => xk, xl=>xl,
                         yi => yi, yj => yj, yk => yk, yl=>yl,
                         zi => zi, zj => zj, zk => zk, zl=>zl,
                         ti => ti, tj => tj, tk => tk, tl=>tl,
                         z1 => z1, z2 => z2, x1 => x1,
                         x2 => x2, y1 => y1, y2 => y2);

int_x1<=to_integer(x1); int_x2<=to_integer(x2);
int_y1<=to_integer(y1); int_y2<=to_integer(y2);
int_z1<=to_integer(z1); int_z2<=to_integer(z2);
end stimulus;
```

## APPENDIX C

```
--VHDL test program for hyperbolic.vhd
--TEST CASE 2

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity test is
end test;

architecture stimulus of test is

component hyperbolic
port(xi,xj,xk,xl,yi,yj,yk,yl,zi,zj,zk,zl,ti,tk,tj,tl: in SIGNED(31 downto 0);
     x1,x2,y1,y2,z1,z2: out SIGNED(31 downto 0));
end component;

signal ti:SIGNED(31 downto 0):=to_signed(86320708,32);
signal tj:SIGNED(31 downto 0):=to_signed(75293013,32);
signal tk:SIGNED(31 downto 0):=to_signed(67335895,32);
signal tl:SIGNED(31 downto 0):=to_signed(78283279,32);

signal xi:SIGNED(31 downto 0):="11111111000110011000111110001011";-- (-15102069)
signal xj:SIGNED(31 downto 0):=to_signed(0,32);
signal xk:SIGNED(31 downto 0):=to_signed(15338349,32);
signal xl:SIGNED(31 downto 0):=to_signed(26566800,32);

signal yi:SIGNED(31 downto 0):=to_signed(21482069,32);
signal yj:SIGNED(31 downto 0):=to_signed(6380000,32);
signal yk:SIGNED(31 downto 0):=to_signed(15338349,32);
signal yl:SIGNED(31 downto 0):=to_signed(0,32);

signal zi:SIGNED(31 downto 0):=to_signed(3683495,32);
signal zj:SIGNED(31 downto 0):=to_signed(25789348,32);
signal zk:SIGNED(31 downto 0):=to_signed(15338349,32);
signal zl:SIGNED(31 downto 0):=to_signed(0,32);

signal x1,x2,y1,y2,z1,z2:SIGNED(31 downto 0):=to_signed(0,32);

signal int_x1,int_x2,int_y1,int_y2,int_z1,int_z2:integer:=0;

begin

s1: hyperbolic port map (xi => xi, xj => xj, xk => xk, xl=>xl,
                         yi => yi, yj => yj, yk => yk, yl=>yl,
                         zi => zi, zj => zj, zk => zk, zl=>zl,
                         ti => ti, tj => tj, tk => tk, tl=>tl,
                         z1 => z1, z2 => z2, x1 => x1,
                         x2 => x2, y1 => y1, y2 => y2);

int_x1<=to_integer(x1); int_x2<=to_integer(x2);
int_y1<=to_integer(y1); int_y2<=to_integer(y2);
int_z1<=to_integer(z1); int_z2<=to_integer(z2);
end stimulus;
```

## APPENDIX D

```
/******************************************************************************
// C program for calculating x,y,z position of mobile given four satellite
// positions and TDOAs to mobile - TEST CASE 1
//*****************************************************************************

#include <iostream.h>
#include <math.h>

main()
{
double ti=67335898; double tk=86023981; double tj=78283279;  double tl=75092320;
double xi=0;        double xk=0;        double xj=-15338349; double xl=-18785564;
double yi=26566800; double yk=6380000;  double yj=15338349;  double yl=18785564;
double zi=0;        double zk=25789348; double zj=15338349;  double zl=0;

cout<<"ti = "<<ti<<endl;      cout<<"tj = "<<tj<<endl;      cout<<"tk = "<<tk<<endl;
cout<<"tl = "<<tl<<endl;      cout<<"xi = "<<xi<<endl;      cout<<"xj = "<<xj<<endl;
cout<<"xk = "<<xk<<endl;      cout<<"xl = "<<xl<<endl;      cout<<"yi = "<<yi<<endl;
cout<<"yj = "<<yj<<endl;      cout<<"yk = "<<yk<<endl;      cout<<"yl = "<<yl<<endl;
cout<<"zi = "<<zi<<endl;      cout<<"zj = "<<zj<<endl;      cout<<"zk = "<<zk<<endl;
cout<<"zl = "<<zl<<endl;

double xji=xj-xi; double xki=xk-xi; double xjk=xj-xk; double xlk=xl-xk;
double xik=xi-xk; double yji=yj-yi; double yki=yk-yi; double yjk=yj-yk;
double ylk=yl-yk; double yik=yi-yk; double zji=zj-zi; double zki=zk-zi;
double zik=zi-zk; double zjk=zj-zk; double zlk=zl-zk;

double rij=abs((100000*(ti-tj))/333564); double rik=abs((100000*(ti-tk))/333564);
double rkj=abs((100000*(tk-tj))/333564); double rkl=abs((100000*(tk-tl))/333564);

double s9 =rik*xji-rij*xki; double s10=rij*yki-rik*yji; double s11=rik*zji-rij*zki;
double s12=(rik*(rij*rij + xi*xi - xj*xj + yi*yi - yj*yj + zi*zi - zj*zj)
           -rij*(rik*rik + xi*xi - xk*xk + yi*yi - yk*yk + zi*zi - zk*zk))/2;

double s13=rkl*xjk-rkj*xlk; double s14=rkj*ylk-rkl*yjk; double s15=rkl*zjk-rkj*zlk;
double s16=(rkl*(rkj*rkj + xk*xk - xj*xj + yk*yk - yj*yj + zk*zk - zj*zj)
           -rkj*(rkl*rkl + xk*xk - xl*xl + yk*yk - yl*yl + zk*zk - zl*zl))/2;

double a= s9/s10; double b=s11/s10; double c=s12/s10; double d=s13/s14;
double e=s15/s14; double f=s16/s14; double g=(e-b)/(a-d); double h=(f-c)/(a-d);
double i=(a*g)+b; double j=(a*h)+c;
double k=rik*rik+xi*xi-xk*xk+yi*yi-yk*yk+zi*zi-zk*zk+2*h*xki+2*j*yki;
double l=2*(g*xki+i*yki+zki);
double m=4*rik*rik*(g*g+i*i+1)-l*l;
double n=8*rik*rik*(g*(xi-h)+i*(yi-j)+zi)+2*l*k;
double o=4*rik*rik*((xi-h)*(xi-h)+(yi-j)*(yi-j)+zi*zi)-k*k;
double s28=n/(2*m);     double s29=(o/m);       double s30=(s28*s28)-s29;
double root=sqrt(s30);        cout<<endl;
int z1=s28+root;              //cout<<"z1 = "<<z1 <<endl;
int z2=s28-root;              cout<<"z2 = "<<z2 <<endl;
int x1=g*z1+h;                //cout<<"x1 = "<<x1 <<endl;
int x2=g*z2+h;                cout<<"x2 = "<<x2 <<endl;
int y1=a*x1+b*z1+c;           //cout<<"y1 = "<<y1 <<endl;
int y2=a*x2+b*z2+c;           cout<<"y2 = "<<y2 <<endl;
}
```

## APPENDIX E

```
//*****************************************************************************
// C program for calculating x,y,z position of mobile given four satellite
// positions and TDOAs to mobile - TEST CASE 2
//*****************************************************************************

#include <iostream.h>
#include <math.h>

main()
{
double ti=86320708;  double tk=67335895; double tj=75293013; double tl=78283279;
double xi=-15102069; double xk=15338349; double xj=0;        double xl=26566800;
double yi=21482069;  double yk=15338349; double yj=6380000;  double yl=0;
double zi=3683495;   double zk=15338349; double zj=25789348; double zl=0;

cout<<"ti = "<<ti<<endl;      cout<<"tj = "<<tj<<endl;      cout<<"tk = "<<tk<<endl;
cout<<"tl = "<<tl<<endl;      cout<<"xi = "<<xi<<endl;      cout<<"xj = "<<xj<<endl;
cout<<"xk = "<<xk<<endl;      cout<<"xl = "<<xl<<endl;      cout<<"yi = "<<yi<<endl;
cout<<"yj = "<<yj<<endl;      cout<<"yk = "<<yk<<endl;      cout<<"yl = "<<yl<<endl;
cout<<"zi = "<<zi<<endl;      cout<<"zj = "<<zj<<endl;      cout<<"zk = "<<zk<<endl;
cout<<"zl = "<<zl<<endl;

double xji=xj-xi; double xki=xk-xi; double xjk=xj-xk; double xlk=xl-xk;
double xik=xi-xk; double yji=yj-yi; double yki=yk-yi; double yjk=yj-yk;
double ylk=yl-yk; double yik=yi-yk; double zji=zj-zi; double zki=zk-zi;
double zik=zi-zk; double zjk=zj-zk; double zlk=zl-zk;

double rij=abs((100000*(ti-tj))/333564); double rik=abs((100000*(ti-tk))/333564);
double rkj=abs((100000*(tk-tj))/333564); double rkl=abs((100000*(tk-tl))/333564);

double s9 =rik*xji-rij*xki; double s10=rij*yki-rik*yji; double s11=rik*zji-rij*zki;
double s12=(rik*(rij*rij + xi*xi - xj*xj + yi*yi - yj*yj + zi*zi - zj*zj)
           -rij*(rik*rik + xi*xi - xk*xk + yi*yi - yk*yk + zi*zi - zk*zk))/2;

double s13=rkl*xjk-rkj*xlk; double s14=rkj*ylk-rkl*yjk; double s15=rkl*zjk-rkj*zlk;
double s16=(rkl*(rkj*rkj + xk*xk - xj*xj + yk*yk - yj*yj + zk*zk - zj*zj)
           -rkj*(rkl*rkl + xk*xk - xl*xl + yk*yk - yl*yl + zk*zk - zl*zl))/2;

double a= s9/s10; double b=s11/s10; double c=s12/s10; double d=s13/s14;
double e=s15/s14; double f=s16/s14; double g=(e-b)/(a-d); double h=(f-c)/(a-d);
double i=(a*g)+b; double j=(a*h)+c;
double k=rik*rik+xi*xi-xk*xk+yi*yi-yk*yk+zi*zi-zk*zk+2*h*xki+2*j*yki;
double l=2*(g*xki+i*yki+zki);
double m=4*rik*rik*(g*g+i*i+1)-l*l;
double n=8*rik*rik*(g*(xi-h)+i*(yi-j)+zi)+2*l*k;
double o=4*rik*rik*((xi-h)*(xi-h)+(yi-j)*(yi-j)+zi*zi)-k*k;
double s28=n/(2*m);     double s29=(o/m);       double s30=(s28*s28)-s29;
double root=sqrt(s30);        cout<<endl;
int z1=s28+root;              //cout<<"z1 = "<<z1 <<endl;
int z2=s28-root;              cout<<"z2 = "<<z2 <<endl;
int x1=g*z1+h;                //cout<<"x1 = "<<x1 <<endl;
int x2=g*z2+h;                cout<<"x2 = "<<x2 <<endl;
int y1=a*x1+b*z1+c;           //cout<<"y1 = "<<y1 <<endl;
int y2=a*x2+b*z2+c;           cout<<"y2 = "<<y2 <<endl;
}
```

**Ralph Bucher** received his B.S. in Engineering Management form University of Missouri-Rolla in 1987 and his M.S. in Computer Engineering from New Jersey Institute of Technology in 2000. He is currently with Lucent Technologies at Whippany, New Jersey. His current interests are in DSL, ethernet and ATM circuits. From 1988 to 1998 he worked at Caterpillar Inc., Peoria, IL.

**Durga Misra** has received his M.S. and Ph.D. degrees both in Electrical Engineering from University of Waterloo, Waterloo, Canada in 1985 and 1988, respec tively. Since 1988 he has been with the department of Electrical and Computer Engineering at New Jersey Institute of Technology where he is now a Professor. In 1997 he was on a research leave at the VLSI Research Department of Bell Laboratories, Lucent Technologies at Murray Hill, New Jersey. His current research interests include low power CMOS circuit design and CMOS device reliability. He is an Associate Editor of IEEE Circuits and Devices Magazine and a senior member of IEEE. He is also a member of the Electrochemical Society, SPIE, and Sigma Xi.