A Synthesizable Low Power VHDL Model of the Exact Solution of Three Dimensional Hyperbolic Positioning System

Ralph Bucher and D. Misra
New Jersey Center for Wireless and Telecommunication
Department of Electrical and Computer Engineering
New Jersey Institute of Technology

ABSTRACT

This paper presents a synthesizable low power VHDL model of a three-dimensional hyperbolic positioning system algorithm. The algorithm derives the equations needed to obtain an exact solution for the three dimensional location of a mobile given the locations of four fixed stations (like a GPS satellite or a base station in a cell) and the signal time of arrival (TOA) from the mobile to each station. The VHDL model of the algorithm was implemented and tested using the IEEE numeric_std package. The model can be easily synthesized for hardware implementation.

1. INTRODUCTION

Many organizations are developing competing products to comply with the FCC's E-911 mandate which requires U.S. cellular carriers to provide location information of phone calls, effective October 2001. The accuracy required is 100 meters or better. Many of these products will implement the well known time difference of arrival (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 the three dimensional position depending on the degree of simplicity desired. In this paper, a more general set of equations needed to locate the three dimensional position of a mobile is presented. These equations will be the basis for implementing a positioning algorithm in C++ and VHDL. The VHDL version will utilize the IEEE numeric_std package so it can be synthesized into an ASIC by anyone seeking a hardware implementation.

2. 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, equation (1) is expanded to four equations when the specific locations of four satellites i, j, k, and l are given. This requirement can be easily met since GPS satellites broadcast their exact locations and at least four satellites are guaranteed to be in the horizon of any location on earth. The satellites also broadcast the times they are at their respective locations so that the signal TOAs can be determined. Even though only three equations are needed to solve for three unknowns, adding a fourth equation greatly simplifies the solutions for x, y, and z.

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

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

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

$$ct_l = R_l = \sqrt{(x_l - x)^2 + (y_l - y)^2 + (z_l - z)^2} \tag{5}$$

Solving the above four equations for the three unknowns results in the following set of equations:

$$z = \frac{N}{2M} \pm \sqrt{\left(\frac{N}{2M}\right)^2 - \frac{O}{M}} \tag{6}$$

$$x = Gz + H \tag{7}$$

$$y = Iz + J \tag{8}$$

$$M = 4R_{ik}^2[G^2 + I^2 + 1] - L^2 \tag{9}$$

$$N = 8R_{ik}^2[G(x_i - H) + I(y_i - J) + z_i] + 2LK \tag{10}$$

$$O = 4R_{ik}^2[(x_i - H)^2 + (y_i - J)^2 + z_i^2] - K^2 \tag{11}$$

$$G = \frac{E - B}{A - D} \tag{12}$$

$$H = \frac{F - C}{A - D} \tag{13}$$

$$I = AG + B \tag{14}$$

$$J = AH + C \tag{15}$$

$$K = R_{ik}^2 + x_i^2 - x_k^2 + y_i^2 - y_k^2 + z_i^2 - z_k^2 + 2x_{ki}H + 2y_{ki}J \tag{16}$$

$$L = 2[x_{ki}G + y_{ki}I + 2z_{ki}] \tag{17}$$

$$A = [\frac{R_{ik}x_{ji} - R_{ij}x_{ki}}{R_{ij}y_{ki} - R_{ik}y_{ji}}] \tag{18}$$

$$B = [\frac{R_{ik}z_{ji} - R_{ij}z_{ki}}{R_{ij}y_{ki} - R_{ik}y_{ji}}] \tag{19}$$

$$C = \{R_{ik}[R_{ij}^2 + x_i^2 - x_j^2 + y_i^2 - y_j^2 + z_i^2 - z_j^2] - R_{ij}[R_{ik}^2 + x_i^2 - x_k^2 + y_i^2 - y_k^2 + z_i^2 - z_k^2]\} / 2[R_{ij}y_{ki} - R_{ik}y_{ji}] \tag{20}$$

$$D = [\frac{R_{kl}x_{jk} - R_{kj}x_{lk}}{R_{kj}y_{lk} - R_{kl}y_{jk}}] \tag{21}$$

$$E = [\frac{R_{kl}z_{jk} - R_{kj}z_{lk}}{R_{kj}y_{lk} - R_{kl}y_{jk}}] \tag{22}$$

$$F = \{R_{kl}[R_{kj}^2 + x_k^2 - x_j^2 + y_k^2 - y_j^2 + z_k^2 - z_j^2] - R_{kj}[R_{kl}^2 + x_k^2 - x_l^2 + y_k^2 - y_l^2 + z_k^2 - z_l^2]\} / 2[R_{kj}y_{lk} - R_{kl}y_{jk}] \tag{23}$$

3. VHDL MODEL

The equations for the *x, y* and *z* position of the mobile was modeled in VHDL. The numeric_std package was used to construct the model, which was readily synthesized into a low power digital circuit. The input signals are the *x, y, z* positions of four GPS satellites, *i, j, k, l* are in meters, and the signal TOAs from the individual satellites to the mobile are 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 meters). Therefore, the TOA range is roughly 6,700,000 to 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 results 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 x 10^{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 increase 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 is used to compute the integer square root of a positive integer represented by a 64-bit vector. 64 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 VHDL model for computing x, y, z position of mobile given four satellite positions and TOAs from satellites to mobile is:

```
--------------------------------------------------------------------------
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;
18<=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);

27<=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:="0100000000000000000000000000000000000000
            00000000000000000000000000";
        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;
```

---

4. 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 (low power mode). A high level schematic was generated using AUTOLOGIC's default component library.

A VHDL test bench representing the real life situation in Fig. 1 was used to validate the model. 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 test bench (not shown) converted the base 10 numbers in Fig. 1 to binary numbers and outputted the *x,y* and *z* position of the mobile in binary and base 10 format.

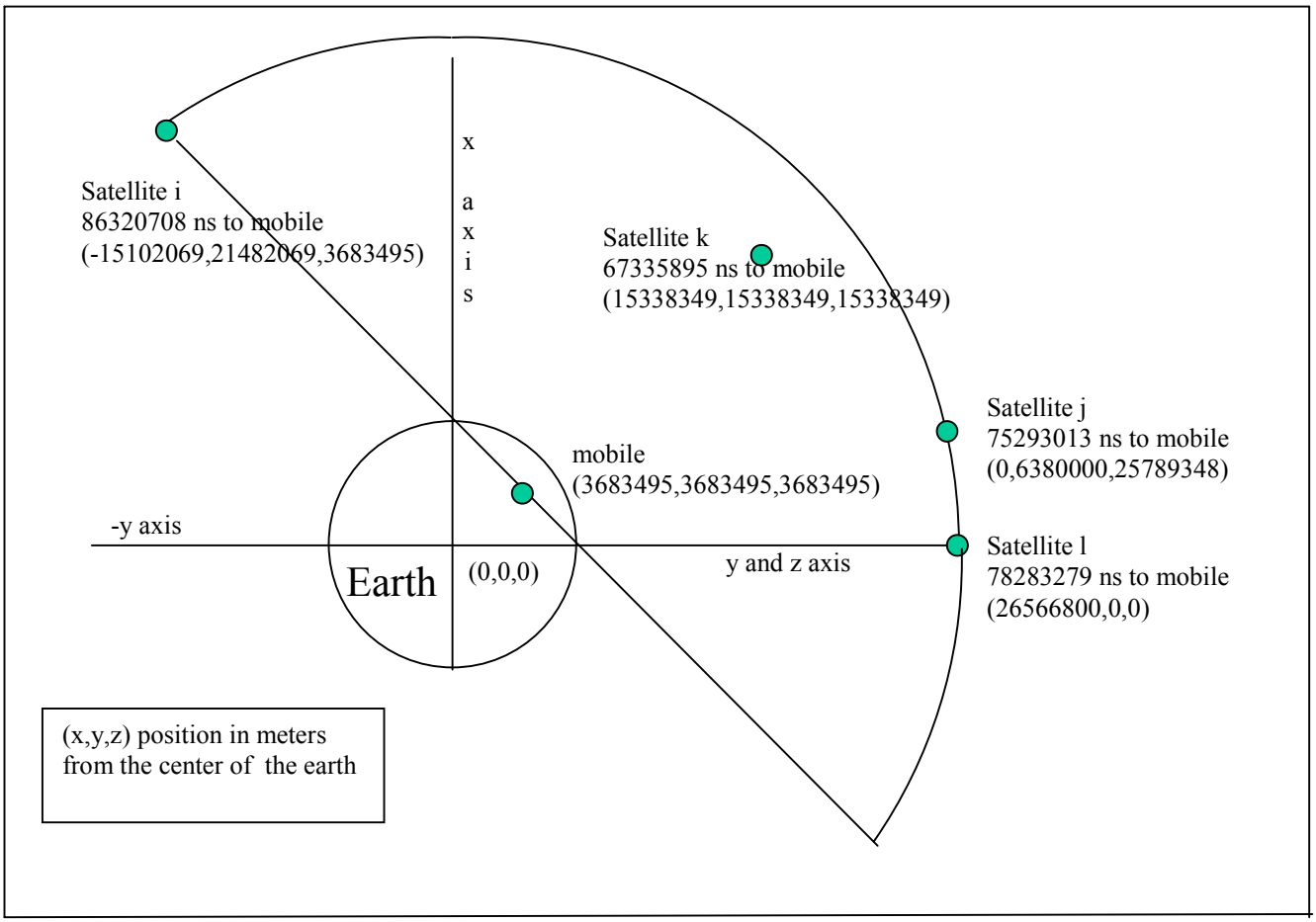

Figure 1

A C++ program using real numbers was used as a benchmark for the accuracy and precision of the VHDL model.

The VHDL simulator displayed the following results:

OUTPUT SIGNAL	VALUE
INT_X1	3683494
INT_Y1	3683495
INT_Z1	3682495
X1	00000000001110000011010010100110
Y1	00000000001110000011010010100111
Z1	00000000001110000011010010100111

Table 1

The C++ program generated the following results:

OUTPUT SIGNAL	VALUE
X1	3683494
Y1	3683495
Z1	3683495

Table 2

5. RESULTS DISCUSSION

The C++ program and VHDL model produced the same results. This means the VHDL model can produce the coordinates as accurate as a GPS positioning system utilizing a general purpose microprocessor with a 32-bit IEEE floating point ALU.

The *x* position was off by one meter for this set of test data. Another set of test data not shown here produced a coordinate which was off by 36 meters due to the satellites being positioned closer to one another. However, this can be corrected by extending the precision beyond the ten decimal points used in the model.

REFERENCES

[1] B.T. Fang, "Simple solutions for a hyperbolic and related position fixes", IEEE Trans. on Aerosp. and Elect. Systems, vol. 26, no. 5, pp. 748-753, Sept 1990.

[2] K.J. Krizman, T.E. Biedka, and T.S. Rappaport, "Wireless position location: fundamentals, implementation strategies, and sources of error", invited paper.