A DIGITAL INTEGRATED INERTIAL NAVIGATION SYSTEM FOR AERIAL VEHICLES

BY

MUHAMMAD IJAZ

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER ENGINEERING

May 2015

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN- 31261, SAUDI ARABIA
DEANSHIP OF GRADUATE STUDIES

This thesis, written by Muhammad Ijaz under the direction of his thesis advisor and approved by his thesis committee, has been presented and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER ENGINEERING**.

Dr. Alaaeldin A. Monem Amin
(Advisor)

Dr. Ahmad Al-Mulhem
Department Chairman

Dr. Moustafa Elshafei
(Member)

Dr. Salam A. Zummo
Dean of Graduate Studies

Dr. Muhammad Y. Mahmoud
(Member)

31/5/15
Date

© Muhammad Ijaz

2015

Dedicated to my beloved

parents, siblings, wife and children

ACKNOWLEDGMENTS

First and foremost, all praise is due to Allah, *subhanahu-wa-ta'ala*, Almighty who gave me the opportunity, courage and patience to complete MS degree at King Fahd University of Petroleum and Minerals, and guided me in every aspect of this work to complete it successfully. May the peace and many blessings of Allah be upon His beloved prophet, Muhammad (S.A.W) whose love is part of our Imaan.

Acknowledgement is due to King Fahd University of Petroleum and Minerals for its financial support and providing world class academic facilities and extra-curricular activities.

I would like to express, with deep and indebted sense of gratitude, my deep appreciation and sincere thanks to my thesis advisor, Dr. Alaaeldin Abdul Monem Amin for his generous support, guidance, continuous encouragement and every possible cooperation throughout my research and preparation of thesis documentation. Working with him, indeed, was a pleasant and learning experience, which I thoroughly enjoyed.

I would also like to extend my sincere thanks and gratitude to other committee members, Dr. Moustafa Elshafei and Dr. Mohammad Y. Mahmoud for their valuable advice and support, suggestions, and critical remarks which helped improve this work.

I would also like to thank Mr. Yazan Muhammad Al-Rawashdeh for his help and support. I am grateful to all the faculty and staff of the Computer Engineering department, who has in one way or the other enriched my academic and research experience at KFUPM. I owe a deep sense of gratefulness to my numerous friends and colleagues whose presence and

moral support made my entire stay at KFUPM enjoyable and fruitful. I would like to sincerely thank them all.

Finally, I would like to express my profound gratitude to my parents, my siblings and all family members for their constant inspiration, incessant prayers, love, support, and encouragement that motivated me to complete this work.

TABLE OF CONTENTS

ACKNOWLEDGMENTS V

TABLE OF CONTENTS VII

LIST OF TABLES IX

LIST OF FIGURES X

LIST OF ABBREVIATIONS XI

ABSTRACT XII

ملخص الرسالة XIII

CHAPTER 1 INTRODUCTION 1

1.1 Navigation 1

1.2 Inertial Navigation Systems (INS) 2
  1.2.1 Stabilized Platform (Gimbaled) INS 2
  1.2.2 Strap Down INS 4

1.3 All-Accelerometer based Integrated Inertial Navigation System 5

1.4 Problem Statement 13

1.5 Objectives 13

1.6 Thesis organization 14

CHAPTER 2 LITERATURE REVIEW 15

2.1 CORDIC 15

2.2 Multipliers 25

2.3 Dividers 28

2.4 Square Rooters 28

2.5 Tangent Inverse 29

CHAPTER 3 PROPOSED HARDWARE DESCRIPTION 31

3.1 Design assumptions 31

3.2 Design inputs and outputs 32

3.3 VHDL design methodology 34

3.4 Hardware design break down 34
- 3.4.1 Request-Acknowledge (*Req-Ack*) Protocol 36
- 3.4.2 Stage 1: Angular Velocity Estimation 37
- 3.4.3 Stage 2 and 3: Computation of omega squares, h_i's and d1 38
- 3.4.4 Stage 4 and 5: Computation of d2, d3 and d3×hi's signals 38
- 3.4.5 Stage 6: Quaternion (q1, q2, q3, q4) Computation 39
- 3.4.6 Stage 7: Directional Cosine Matrix 41
- 3.4.7 Stage 8: Square Rooter 42
- 3.4.8 Stage 9: Orientation of the aerial vehicle 43
- 3.4.9 Individual components used in different stages of pipeline 44

CHAPTER 4 RESULTS AND DISCUSSIONS 57

4.1 Verification of VHDL generated results 57

4.2 Synthesis results 61

CHAPTER 5 CONCLUSIONS AND FUTURE WORK 66

5.1 Contributions 66

5.2 Future work 67

REFERENCES 69

VITAE 75

LIST OF TABLES

Table 3.1 : Radix-4 modified Booth multiplier digit encoding 45
Table 4.1 : Synthesis results of the IINS processor 61

LIST OF FIGURES

Figure 1.1: Gimbaled INS platform [1] 3
Figure 1.2: Strap Down INS fitted to an aero plane [2] 4
Figure 1.3: Diamond shaped all-accelerometer based IMU 6
Figure 2.1: Data path of CORDIC processor. 17
Figure 3.1: IINS processor input and output ports 33
Figure 3.2: Mapping of design issues in algorithm and architecture space. 34
Figure 3.3: Different stages of the IINS hardware. 35
Figure 3.4: Timing diagram of *Req-Ack* protocol 36
Figure 3.5: Data path of the angular velocity estimation module 37
Figure 3.6: Data path of the quaternion stage 40
Figure 3.7: Data path of the DCM stage 42
Figure 3.8: FSM controller of the radix-4 Booth multiplier 47
Figure 3.9: Data path of radix-4 Booth multiplier 48
Figure 3.10: Data path of the SRT binary divider 50
Figure 3.11: Data path of the square rooter using CORDIC 53
Figure 3.12: Data path of the CORDIC-based tangent inverse function evaluator. 56
Figure 4.1: Comparison of angular velocities generated by VHDL and MATLAB. 58
Figure 4.2: Comparison of quaternion parameters generated by VHDL and MATLAB . 59
Figure 4.3: Orientation of the vehicle generated by the VHDL model and MATLAB. ... 60
Figure 4.4: Clock period w.r.t no. of bits in input operands 62
Figure 4.5: Input -to- output latency w.r.t no. of bits in input operands 63
Figure 4.6: Area of the proposed processor for IINS w.r.t. no. of bits in input operands 64
Figure 4.7: Area- latency product w.r.t. no. of bits in input operands. 65

LIST OF ABBREVIATIONS

IINS : Integrated Inertial Navigation System

IMU : Inertial Measurement Unit

CORDIC : Co-Ordinate Rotational Digital Computer

CoG : Center of Gravity

DCM : Directional Cosine Matrix

ABSTRACT

Full Name : Muhammad Ijaz
Thesis Title : A Digital Integrated Inertial Navigation System For Aerial Vehicles
Major Field : Computer Engineering
Date of Degree : May, 2015

Recent aerial vehicles are typically equipped with an Inertial Navigation System (INS) which is used to keep track of orientation, velocity and position of the aerial vehicle. The INS uses measurements from sensors (like accelerometers and gyroscopes) together with the knowledge of initial position and velocity to compute the current position and velocity of the vehicle. An all-accelerometer based INS uses only tri-axial accelerometers as sensors and involves complex computations. In this work, an all-accelerometer based INS has been designed as a dedicated Application Specific Integrated Circuit (ASIC) hardware to perform these complicated computations. The architecture, and design details of the proposed ASIC for INS have been provided. This hardware has been designed using a combination of pipeline and iterative approaches to make it suitable for the guidance of high speed aerial vehicles (e.g. space shuttles and missiles) while having the minimum possible area. A working parameterized VHDL model of this system together with its test bench have been developed. Further, the VHDL model has been synthesized using CADENCE Encounter® RTL compiler with 90 nm Digital Standard Cell library. A full evaluation of the expected speed, latency, and area requirements for different size input operands is performed. Synthesis results show that the proposed INS processor can operate at 1 GHz frequency with input-to-output latency of 54 clock cycles and an area of 492,962 μ^2 for 16 bit input operands.

ملخص الرسالة

الاسم الكامل: محمد إعجاز

عنوان الرسالة: نظام رقمي متكامل لملاحة المركبات الجوية بالقصور الذاتي

التخصص: هندسة الحاسب الآلي

تاريخ الدرجة العلمية: مايو 2015

تزود المركبات الجوية الحديثة عادة بنظام ملاحة بالقصور الذاتي (INS) والذي يستعمل لتتبع توجه ، وسرعة ، و موقع المركبة الجوية. و يقوم ال INS بحساب سرعة المركبة الهوائية و موقعها الحالي باستخدام حسابات مركبة تعتمد على قياسات من أجهزة استشعار معينة (مثل أجهزة قياس التسارع و الجيروسكوبات) جنبا إلى جنب مع معرفة موقع البدء و سرعة البداية. وتستخدم نظم INS والتي تعتمد فقط على أجهزة قياس التسارع المحاور الثلاثة لوضع أجهزة قياس التسارع عليها و إجراء عدد من العمليات الحسابية المركبة على القياسات القادمة من المحاور الثلاثة. في هذا البحث قمنا بتصميم أحد هذه ال INS كدائرة متكاملة عالية الكثافة للقيام بإجراء الحسابات المطلوبة لخدمة هذا التطبيق الخاص. كما قمنا هنا بتوفير وشرح هيكلية هذا التصميم وتفاصيله. وفي هذا التصميم تم استخدام خليط من تقنيات ال pipeline وال iterative ليمكن الإستفادة منه في حالات المركبات الجوية فائقة السرعة مثل المكوكات والصواريخ الفضائية. وقد تم تطوير نموذج عمل VHDL لهذا النظام جنبا إلى جنب وإجراء الإختبارات اللازمة له. علاوة على ذلك فقد تم توليف نموذج VHDL باستخدام Encounter® RTL بتقنية 90 نانومتر مع مكتبة الخلايا القياسية الرقمية. وقد تم إجراء تقييم كامل لتوليفة هذا النموذج من حيث السرعة، والعطلة ، و المساحة المتوقعة لأحجام مختلفة من مدخلات النظام. وأظهرت النتائج أن معالج ال INS المقترح يمكنه العمل بسرعة 1 غيغا هيرتز مع عطلة (مدخلات إلى مخرجات) 54 دورة (clock cycles) وبمساحة قدرها (492,962 μ^2) في حالة استخدام 16 بت كحجم للمدخلات.

CHAPTER 1

INTRODUCTION

1.1 Navigation

The ability to move between two points is known as navigation. There are five basic types of navigation; Dead Reckoning, Celestial Navigation, Pilotage, Inertial Navigation, and Radio Navigation. The Pilotage navigation is very old and is based on recognizing the land marks to identify the current position of any object. The Dead Reckoning type of navigation is based on the knowledge of starting point, and some sort of heading along with some estimate of speed. The Celestial navigation uses the knowledge of time and angle between the visible horizon and celestial objects like sun, moon, etc. Whereas, the Radio navigation relies on radio-frequency sources at known locations. The Inertial Navigation is Dead Reckoning type of navigation which is used to keep track of orientation, velocity and position of any object/vehicle using motion sensors (gyroscopes and accelerometers) without making use of any external references. In an inertial navigation system, initial position and velocity of the vehicle are provided as input and then information from the sensors is integrated to get the current velocity and position of the vehicle. It is immune against jamming and weather changes and is considered as self-contained.

1.2 Inertial Navigation Systems (INS)

An Inertial Navigation System (INS) is a navigational system used to keep track of orientation, velocity and position of aerial vehicle in which it is installed using inertial sensors like accelerometers and gyroscopes. An INS typically contains dedicated electronics, an Inertial Measurement Unit (IMU), and a computing unit. IMU contains a number of accelerometers and gyroscopes which are fastened to a common frame. The body linear/rotational motion can be measured using a number of linear accelerometers or gyroscopes.

Applications of INS include, but are not limited to, aircraft guidance systems, missile guidance systems, unmanned aerial vehicles guidance, submarines and ships guidance, space shuttles guidance, and drones surveillance and guidance systems. The INS has the advantage of being self-contained, inherently stealthy, and immune against signal jamming. There are two main categories of Inertial Navigation Systems namely Gimbaled or stabilized platforms and Strap down INS.

1.2.1 Stabilized Platform (Gimbaled) INS

The main type of INS is Gimbaled which had been used in the initial applications of INS systems. In Gimbaled systems, the sensors are fixed to a stabilized platform to isolate them from the vehicle's rotational motion. These type of systems are still being used in applications (like ships and submarines) which require very accurate navigation data. As depicted in Figure 1.1, a minimum of three gimbals are needed to isolate the sensors from the vehicle's rotational motion in 3D space, labeled as roll, pitch, and yaw (Azimuth) axes[1].

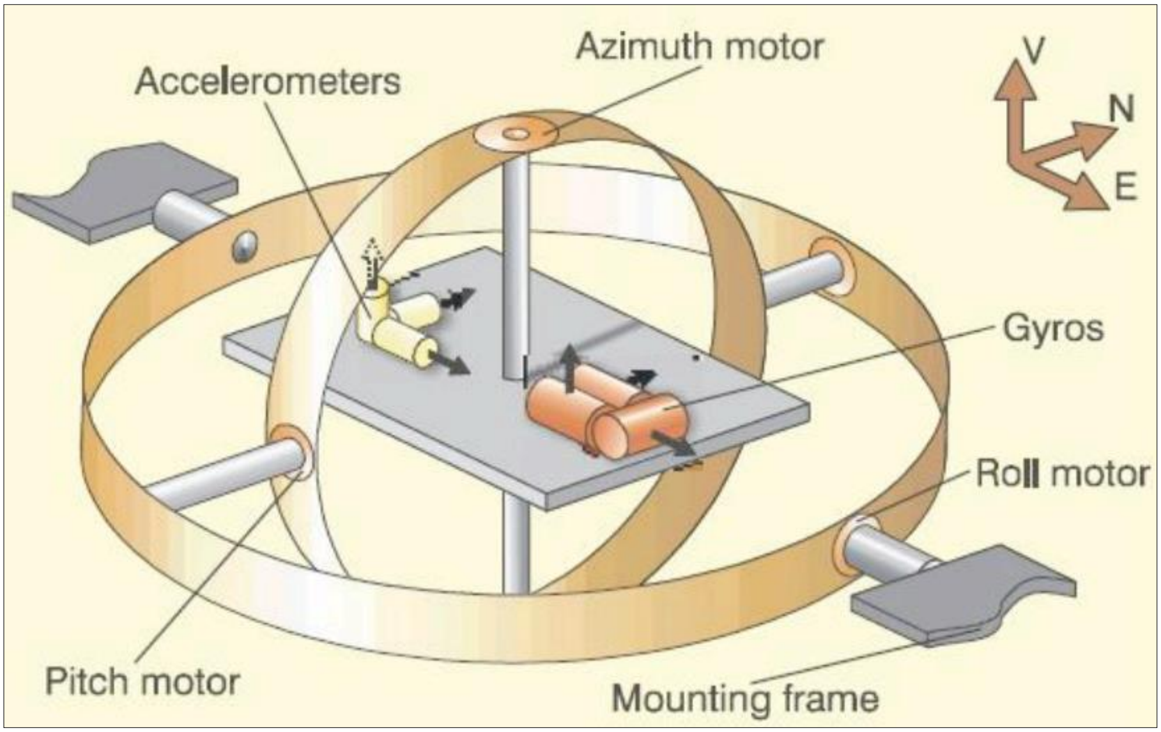


Figure 1.1: Gimbaled INS platform [1]

A mechanism, consisting of gimbals and torque servos, is used to cancel out the rotation of stable platform on which the inertial sensors are mounted. The basic principle of stabilized platform is the cancellation of relative orientation with respect to the inertial frame. Being highly sophisticated and more accurate approach than the strapdown one, it is still used in many vehicles requiring high navigation accuracy such as ships.

1.2.2 Strap Down INS

Second type of INS is called strap down where sensors are "strapped down or" rigidly attached to the body of the aerial vehicle as shown in Figure 1.2 [2]. This type of inertial navigation system has removed the most mechanical complexity from the gimbaled platform systems and is very popular amongst modern systems. The potential benefits of this approach are lower cost, reduced size, and greater reliability as compared to stabilized platform systems.

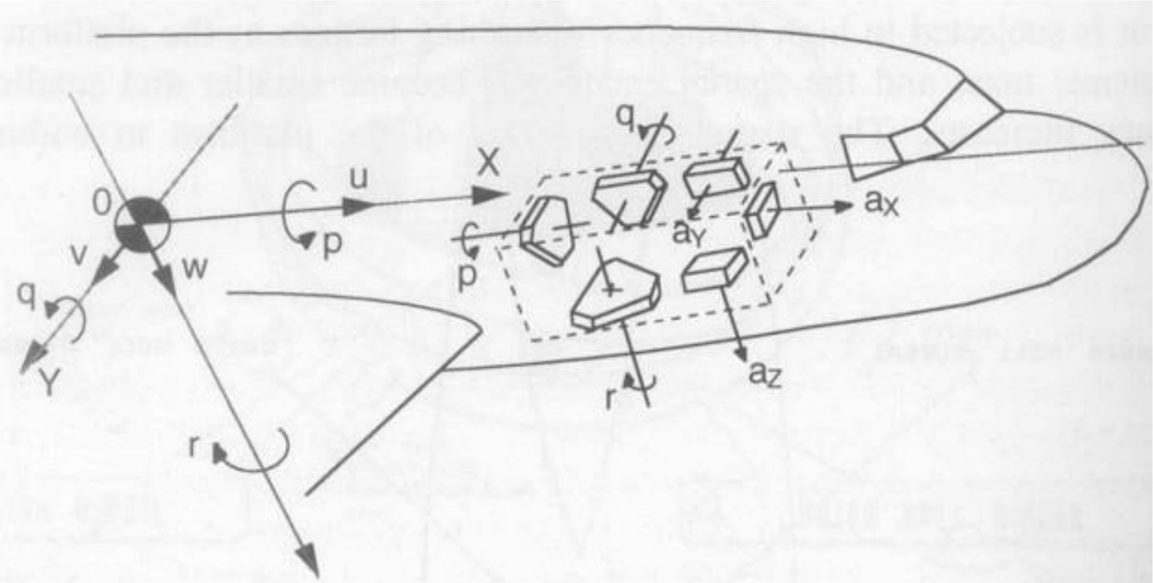


Figure 1.2: Strap Down INS fitted to an aero plane [2]

In order to retrieve the vehicle dynamics, Inertial Measurement Units (IMUs) are generally used in aerial vehicles. Most IMUs are comprised of a number of accelerometers, gyros, and magnetometers that may vary according to the application. The IMU unit is one block within the Inertial Navigation System (INS) which is used to measure the angular accelerations and rotations of the vehicle.

1.3 All-Accelerometer based Integrated Inertial Navigation System

The feasibility of designing an all-accelerometer based IMU using only linear accelerometers' measurements to compute the linear/angular accelerations, and the angular velocity of a rigid body was investigated in [3]. Using differential mode output of the linear accelerometers in certain configuration makes it possible to find the angular acceleration of the body to which they are attached.

Diamond configuration of the accelerometers is one popular configuration in which two linear accelerometers are equally separated about a point in three perpendicular directions, i.e. one pair per axis. The differential output of these pairs of accelerometers is then fed into a Kalman Filter (KF) or the like to estimate the body angular velocities from the noisy linear accelerometers' measurements which, in turn, can be integrated to find the position of aerial vehicle. In the IMU reported in [4], only two pairs of linear tri-axial accelerometers were used in the Y and Z directions having a total number of 12 accelerometers. The IMU reported in [5], however, improved this technique by adding another pair of tri-axial accelerometers on x-axis.

In the IMU (Inertial Measurement Unit) presented in [5], three pairs of linear tri-axial accelerometers were used in the X, Y and Z directions for a total number of 18 accelerometers as shown in Figure 1.3.

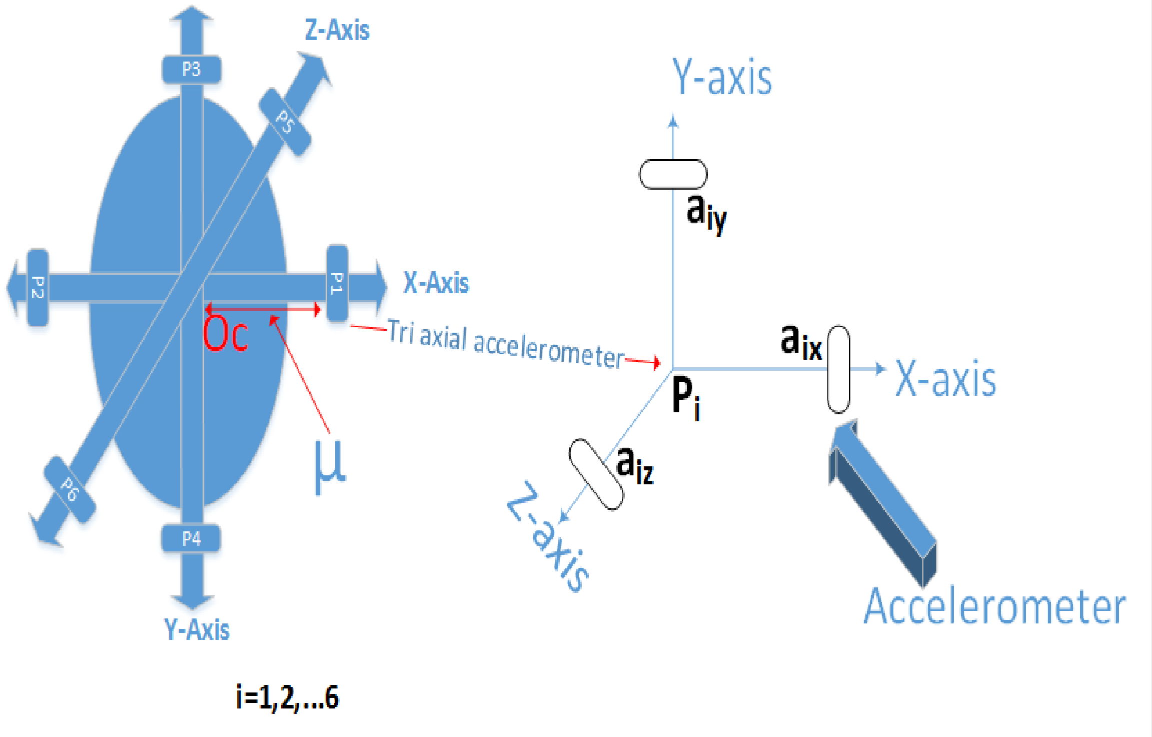


Figure 1.3: Diamond shaped all-accelerometer based IMU

Figure 1.3 shows the all-accelerometer based IMU for the IINS reported in [5]. Points P1 to P6 represent 3-pairs of tri-axial accelerometers on the X, Y, and Z axes. A total of 18 accelerometers are used (3-pairs of tri-axial accelerometers means a total of 18 accelerometers). These accelerometers are symmetrically placed around point *Oc* (center of gravity of the moving body) at some distance μ[1]. Such Inertial Navigational System consists of the following modules:

1. Angular Velocities Estimation module $[\Omega(t)] = [\omega_1, \omega_2, \omega_3]'$.

---

[1] $1cm \leq \mu \leq 10$ cm

Given the accelerometers' measurements at points P1-P6 (Figure 1.3), an angular velocities estimation module is periodically activated every sampling period Ts[2] to estimate the angular velocities. To find angular velocities from the accelerometers' measurements, the *angvel*[3] module integrates the differential outputs of the 3 pairs with respect to time.

2. Updating the quaternion equation. After calculating the angular velocities of the vehicle, equation (1.2) is updated at each sampling period Ts, assuming initial condition as $q(t=0)=[0,0,0,1]'$ where q (t) is represented by (1.2) and this update operation is performed within *quatern* module.

$$\dot{q}=\frac{1}{2}q\Omega \tag{1.1}$$

$$q(t)=quatern(\,q(t-1),\Omega(t),t\,) \tag{1.2}$$

3. Obtaining the Directional Cosine Matrix (C=*DCM* (q(t))) .
4. Obtaining the aircraft attitude from the quaternion by converting quaternions to Euler angles [$\theta(t),\varphi(t),\psi(t)=attitude(q(t))$].
5. Finding the body acceleration, velocity, position, and position of the CoG.

For simplicity, the differential output of each pair of accelerometers located in the X, Y, and Z directions are used, and the aerial vehicle's dynamics are computed in five stages as follows:

---

[2] $1\ ms \leq \text{Ts} \leq 5\ ms$

[3] $\Omega(t)=angvel(A^1(t),A^2(t),A^3(t),A^4(t),A^5(t),A^6(t),\Omega(t-1))$

Stage 1: Acceleration measurement at point P1 (Figure 1.3) is given by:

$$A^1 = A + g_v + \dot{\Omega} \times (\mu \vec{i}) + \Omega \times (\Omega \times (\mu \vec{i})) \tag{1.3}$$

Where,

$$A^1 = \begin{bmatrix} A_{1x} \\ A_{1y} \\ A_{1z} \end{bmatrix} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} + Q \begin{bmatrix} 0 \\ 0 \\ g_0 \end{bmatrix} + \begin{bmatrix} 0 & -\dot{\Omega}_z & \dot{\Omega}_y \\ \dot{\Omega}_z & 0 & -\dot{\Omega}_x \\ -\dot{\Omega}_y & \dot{\Omega}_x & 0 \end{bmatrix} \begin{bmatrix} \mu \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -\Omega_z^2 - \Omega_y^2 & \Omega_x \Omega_y & \Omega_x \Omega_z \\ \Omega_x \Omega_y & -\Omega_z^2 - \Omega_x^2 & \Omega_z \Omega_y \\ \Omega_x \Omega_z & \Omega_z \Omega_y & -\Omega_y^2 - \Omega_x^2 \end{bmatrix} \begin{bmatrix} \mu \\ 0 \\ 0 \end{bmatrix} \tag{1.4}$$

Where, Q is a transformation from inertia to body axes, and $\begin{bmatrix} a_x & a_y & a_z \end{bmatrix}'$ is the body acceleration at the CoG.

Similarly, acceleration measured at point P2 is given by:

$$A^2 = \begin{bmatrix} A_{2x} \\ A_{2y} \\ A_{2z} \end{bmatrix} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} + Q \begin{bmatrix} 0 \\ 0 \\ g_0 \end{bmatrix} + \begin{bmatrix} 0 & -\dot{\Omega}_z & \dot{\Omega}_y \\ \dot{\Omega}_z & 0 & -\dot{\Omega}_x \\ -\dot{\Omega}_y & \dot{\Omega}_x & 0 \end{bmatrix} \begin{bmatrix} -\mu \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -\Omega_z^2 - \Omega_y^2 & \Omega_x \Omega_y & \Omega_x \Omega_z \\ \Omega_x \Omega_y & -\Omega_z^2 - \Omega_x^2 & \Omega_z \Omega_y \\ \Omega_x \Omega_z & \Omega_z \Omega_y & -\Omega_y^2 - \Omega_x^2 \end{bmatrix} \begin{bmatrix} -\mu \\ 0 \\ 0 \end{bmatrix} \tag{1.5}$$

The acceleration difference of points P1 and P2 is then given by:

$$A^1 - A^2 = \begin{bmatrix} 0 & -\dot{\Omega}_z & \dot{\Omega}_y \\ \dot{\Omega}_z & 0 & -\dot{\Omega}_x \\ -\dot{\Omega}_y & \dot{\Omega}_x & 0 \end{bmatrix} \begin{bmatrix} 2\mu \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -\Omega_z^2 - \Omega_y^2 & \Omega_x \Omega_y & \Omega_x \Omega_z \\ \Omega_x \Omega_y & -\Omega_z^2 - \Omega_x^2 & \Omega_z \Omega_y \\ \Omega_x \Omega_z & \Omega_z \Omega_y & -\Omega_y^2 - \Omega_x^2 \end{bmatrix} \begin{bmatrix} 2\mu \\ 0 \\ 0 \end{bmatrix} \tag{1.6}$$

Which yields:

$$\begin{aligned} a_{1x} - a_{2x} &= 2\mu(-\Omega_z^2 - \Omega_y^2) \\ a_{1y} - a_{2y} &= 2\mu(\dot{\Omega}_z + \Omega_x\Omega_y) \\ a_{1z} - a_{2z} &= 2\mu(-\dot{\Omega}_y + \Omega_x\Omega_z) \end{aligned} \tag{1.7}$$

Likewise, the differential output of accelerometers at points P3 and P4 is given by:

$$\begin{aligned} a_{3x} - a_{4x} &= 2\mu(-\dot{\Omega}_z + \Omega_x\Omega_y) \\ a_{3y} - a_{4y} &= 2\mu(-\Omega_z^2 - \Omega_x^2) \\ a_{3z} - a_{4z} &= 2\mu(\dot{\Omega}_x + \Omega_z\Omega_y) \end{aligned} \tag{1.8}$$

Similarly, the differential accelerations of P5 and P6 is given by:

$$\begin{aligned} a_{5x} - a_{6x} &= 2\mu(\dot{\Omega}_y + \Omega_x\Omega_z) \\ a_{5y} - a_{6y} &= 2\mu(-\dot{\Omega}_x + \Omega_z\Omega_y) \\ a_{5z} - a_{6z} &= 2\mu(-\Omega_x^2 - \Omega_y^2) \end{aligned} \tag{1.9}$$

The following equations yield the angular accelerations which are then integrated to estimate the angular velocity ($\omega_1,\ \omega_2,\ \omega_3$) of the body.

$$\begin{aligned} \dot{\omega}_1 &= \frac{1}{4\mu}(a_{3z} - a_{4z} + a_{6y} - a_{5y}) \\ \dot{\omega}_2 &= \frac{1}{4\mu}(a_{5x} - a_{6x} + a_{2z} - a_{1z}) \\ \dot{\omega}_3 &= \frac{1}{4\mu}(a_{1y} - a_{2y} + a_{4x} - a_{3x}) \end{aligned} \tag{1.10}$$

A discretized solution of (1.10) is given by:

$$\begin{aligned} \frac{\omega_1(t) - \omega_1(t-1)}{T_s} &= \frac{1}{4\mu}(a_{3z} - a_{4z} + a_{6y} - a_{5y}) \\ \frac{\omega_2(t) - \omega_2(t-1)}{T_s} &= \frac{1}{4\mu}(a_{5x} - a_{6x} + a_{2z} - a_{1z}) \\ \frac{\omega_3(t) - \omega_3(t-1)}{T_s} &= \frac{1}{4\mu}(a_{1y} - a_{2y} + a_{4x} - a_{3x}) \end{aligned} \tag{1.11}$$

Where a_{ix} are the measurements of accelerometers at points P1-P6 along the X, X, and Z axes. While Ts is the sampling period.

Stage 2: After calculating the angular velocities, quaternion parameterization is used to find the attitude of the aerial vehicle.

$$\dot{q} = \frac{1}{2} q(\omega_1 i + \omega_2 j + \omega_3 k) \tag{1.12}$$

Where Euler parameters q_i are the components of a quaternion defined by:

$$q = q_1 i + q_2 j + q_3 k + q_4 \tag{1.13}$$

These quaternion parameters are extensions of the complex numbers, and their unit vectors satisfy the following relations:

$$\begin{gathered} i^2 = j^2 = k^2 = -1 \\ ij = -ji = k \\ jk = -kj = i \\ ki = -ik = j \end{gathered} \tag{1.14}$$

In a vector-matrix form, equation (1.12) can be written as follows

$$\dot{q} = \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \dot{q}_4 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 0 & \omega_3 & -\omega_2 & \omega_1 \\ -\omega_3 & 0 & \omega_1 & \omega_2 \\ -\omega_2 & -\omega_1 & 0 & \omega_3 \\ -\omega_1 & -\omega_2 & -\omega_3 & 0 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} \tag{1.15}$$

Then orientation of the aircraft w.r.t. an initial condition can be obtained by integrating the above relation. Using the integration method for quaternion and directional cosine matrix described in [6] equation (1.15) can be integrated as follows.

Define:

$$h_0 = Ts(\omega_1^2 + \omega_2^2 + \omega_3^2)^{1/2}\,; \qquad h_i = \omega_i Ts\,;\ where:\ i = 1,2,3. \tag{1.16}$$

$$d_1 = \frac{h_o^2}{16}; \quad d_2 = \frac{1-d_1}{1+d_1}; \quad d_3 = \frac{1}{2(1+d_1)} \tag{1.17}$$

Then:

$$\begin{aligned}
q_1(t) &= d_2 \times q_1(t-1) + d_3 \times h_3 \times q_2(t-1) - d_3 \times h_2 \times q_3(t-1) + d_3 \times h_1 \times q_4(t-1) \\
q_2(t) &= d_2 \times q_2(t-1) - d_3 \times h_3 \times q_1(t-1) + d_3 \times h_1 \times q_3(t-1) + \mathrm{d}_3 \times h_2 \times q_4(t-1) \\
q_3(t) &= d_2 \times q_3(t-1) + d_3 \times h_3 \times q_4(t-1) - d_3 \times h_1 \times q_2(t-1) - \mathrm{d}_3 \times h_2 \times q_1(t-1) \\
q_4(t) &= d_2 \times q_4(t-1) - d_3 \times h_3 \times q_3(t-1) - d_3 \times h_1 \times q_1(t-1) - \mathrm{d}_3 \times h_2 \times q_2(t-1)
\end{aligned} \tag{1.18}$$

From these values of the quaternion components direction cosine matrix is constructed as shown in stage 3.

Stage 3: Once quaternions are calculated, the direction cosine matrix is constructed from the quaternion components as shown below.

$$C = \begin{bmatrix} q_1^2 - q_2^2 - q_3^2 + q_4^2 & 2(q_1q_2 - q_3q_4) & 2(q_3q_1 + q_2q_4) \\ 2(q_1q_2 + q_3q_4) & -q_1^2 + q_2^2 - q_3^2 + q_4^2 & 2(q_2q_3 - q_1q_4) \\ 2(q_3q_1 - q_2q_4) & 2(q_2q_3 + q_1q_4) & -q_1^2 - q_2^2 + q_3^2 + q_4^2 \end{bmatrix} \tag{1.19}$$

$$\mathrm{C}_o = 1 - 8q_1q_2q_3q_4$$

Stage 4: From this DCM, Euler angels (the orientation of the aircraft) are calculated as follows:

$$\begin{aligned} \theta_1 &= \tan^{-1}\left(\frac{-C_{23}}{C_{33}}\right); \qquad where: -180 \le \theta_1 \le 180; \\ \theta_2 &= \tan^{-1}\left(\frac{C_{13}}{\sqrt{C_o}}\right); \qquad where: -90 \le \theta_2 \le 90; \\ \theta_3 &= \tan^{-1}\left(\frac{C_{12}}{C_{11}}\right); \qquad where: -180 \le \theta_3 \le 180; \end{aligned} \tag{1.20}$$

The following method may be used to evaluate $\phi = \tan^{-1}(y/x) = \tan^{-1}(y, x)$.

1-Find:

$$\phi_o = \begin{cases} \tan^{-1}\left(|y|, |x|\right) & if\ |y| < |x| \\ \tan^{-1}\left(\left|\frac{y}{x}\right| - 1, \left|\frac{y}{x}\right| + 1\right) & if\ |y| \ge |x| \end{cases} \tag{1.21}$$

2-Determine the correct angle ϕ depending upon the quadrant it is located in as follows:

$$\phi = \begin{cases} \phi_0\ ; & \text{when x} \ge 0;\ \text{y} \ge 0 \\ -\phi_0\ ; & when\ \text{x} \ge 0;\ \text{y} < 0 \\ \pi - \phi_0\ ; & when\ \text{x} < 0;\ \text{y} \ge 0 \\ \phi_0 - \pi\ ; & when\ x < 0;\ \text{y} < 0 \end{cases} \tag{1.22}$$

Stage 5:

Estimate the aircraft inertial position, velocity, and acceleration from the body measurements using the directional cosine matrix.

1.4 Problem Statement

As clear from equations (1.1)-(1.22), an all-accelerometer based IINS involves many tedious computations. So, a dedicated application specific hardware is required to perform all these complicated computations. This thesis work attempts to design an application specific hardware for all-accelerometer based IINS reported in [5] which will be used (along with other circuitry of the IINS) to guide the aerial vehicle.

1.5 Objectives

The objective of this work is to investigate the possibility of designing an ASIC IINS processor which performs computations involved in equations (1.1)-(1.22), evaluate its merits and recommend strategies for future implementations. This objective includes the following tasks:

1. Describe an architecture and design details of the proposed hardware for the IINS system. A pipelined architecture will be investigated for very high speed aerial vehicles (e.g. space shuttles and missiles).
2. Develop a working parameterized VHDL model for this system and verify results generated from this model against known results produced using MATLAB.
3. Synthesize the VHDL model and evaluate merits of the resulting implementation, e.g. area, speed, and latency for different input precisions.

1.6 Thesis organization

This thesis consists of five chapters. First chapter gives an introduction of the Integrated Inertial Navigation System (IINS) and defines the objectives of this work.

Chapter two reviews techniques used to build different components needed for the IINS processor including multipliers, dividers, square rooters, and trigonometric function evaluators.

Chapter three describes the developed IINS hardware.

In chapter four, we present synthesis results and discuss their implications. Whereas, in chapter five we make conclusions and suggest possible future work.

CHAPTER 2

LITERATURE REVIEW

This chapter provides literature review of the algorithms and techniques used to realize different components required to implement the IINS hardware.

2.1 CORDIC

The Co-Ordinate Rotation DIgital Computer (CORDIC) algorithm was introduced by Volder in 1959 [7] and later on generalized and unified by Walther [8]. The unified algorithm computes trigonometric, hyperbolic, exponential and logarithmic functions, as well as, multiplication, division and square root. CORDIC is an attractive algorithm because it can compute most mathematical functions using basic operations of the form of $a \pm b \times 2^{-i}$ using simple hardware.

At the time of CORDIC introduction, multipliers were very expensive. Hence, CORDIC was an attractive way to evaluate elementary functions instead of using polynomial techniques. CORDIC was designed to be a special-purpose digital computer for real-time airborne computation. It was proposed by Volder to solve trigonometric relationships involved in plane coordinate rotation and conversion from rectangular to polar coordinates. At that time, compared to the analog devices, the basic operation of CORDIC can be functionally described as the digital equivalent of an analog resolver. Originally, CORDIC was programmed to solve either set of the following equations:

$$\begin{aligned} y_b &= K\left(y_a cos\theta + x_a sin\theta\right) \\ x_b &= K\left(x_a cos\theta - y_a sin\theta\right) \end{aligned} \tag{2.1}$$

Or

$$\begin{aligned} R &= K\sqrt{(x^2 + y^2)} \\ \theta &= tan^{-1}(y/x) \end{aligned} \tag{2.2}$$

Where K is a constant.

There are two modes of operation in CORDIC namely Rotation and Vectoring. In Rotation operation equation (2.1) is used to calculate the rotated coordinates of a point around origin where the amount of rotation is equal to the input angle θ, whereas in Vectoring mode equation (2.2) is used to calculate the magnitude and angle of a given vector. In rotation operation, initial coordinates of a two dimensional vector and a rotation angle are given as input; and the output is the rotated components of that vector. While in vectoring operation, the coordinate components of a two dimensional vector are provided as input while the magnitude and the angle of the original vector with the x-axis are produced as output.

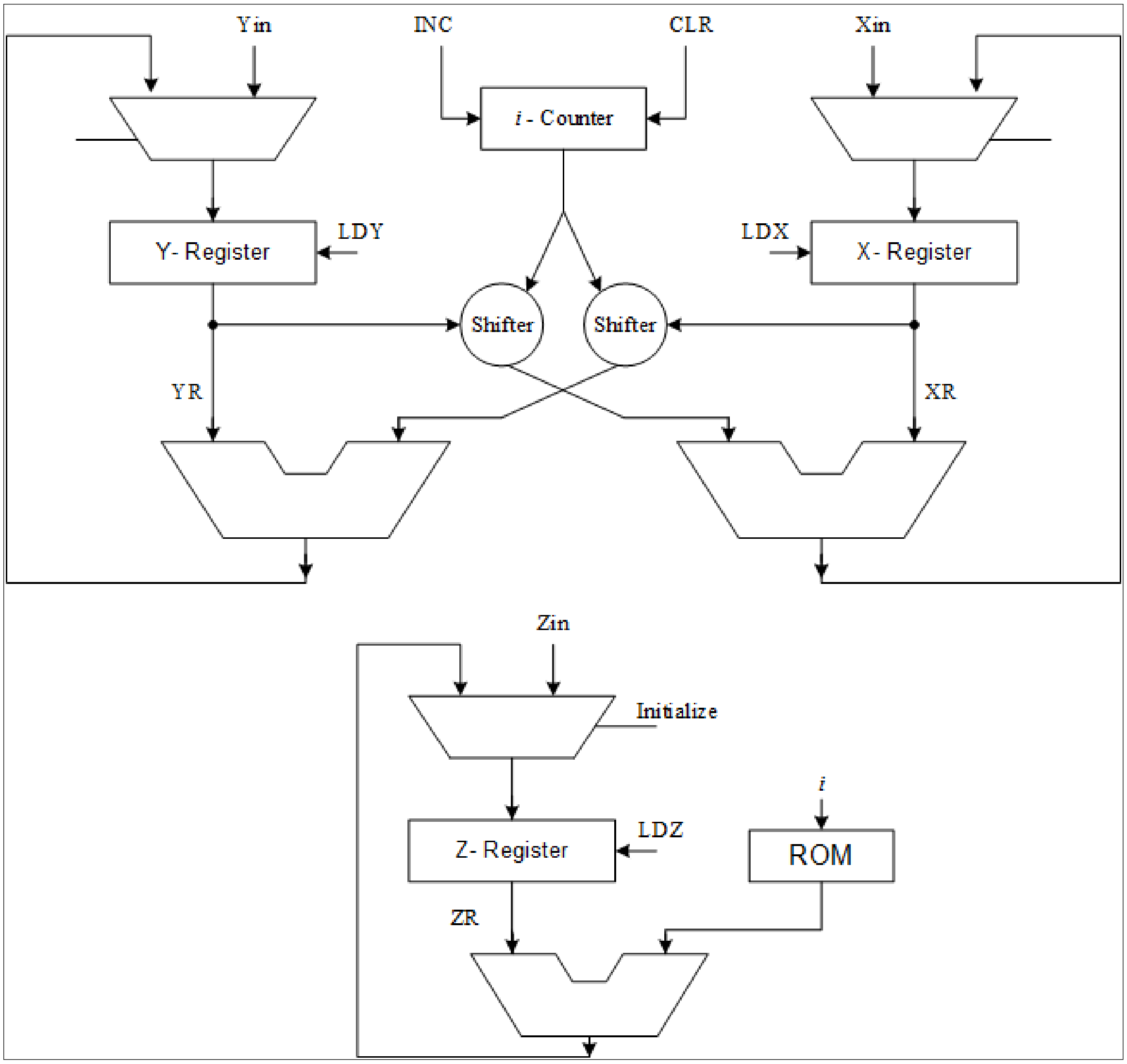


Figure 2.1: Data path of CORDIC processor.

The basic computing method used in both rotation and vectoring operations is a step-by-step sequence of micro rotations which results in an overall rotation by a given angle as in the rotation operation, or results in zeroing the final angle of the vector as in vectoring operation. These micro rotation angles are chosen such that the computations needed are only shift and add operations. The micro rotations are performed recursively, where the number of iterations required is equal to the number of significant bits that represent the rotated components.

Walther [8] generalized CORDIC to be used in other coordinate systems such that instead of rotating a vector along a circular curve, the vector can be rotated along a line or a hyperbola, in circular, linear, and hyperbolic coordinate systems, respectively. Walther came up with a set of unified equations that describes the coordinate components of the rotated vector. These equations are parameterized in terms of the coordinate system as shown below.

$$\begin{aligned} x_{i+1} &= x_i - m d_i y_i 2^{-i} \\ y_{i+1} &= y_i + d_i x_i 2^{-i} \\ z_{i+1} &= z_i - d_i e^i \end{aligned} \tag{2.3}$$

Where $m = \begin{cases} 1 & \textit{for circular coordinates} \\ 0 & \textit{for linear coordinates} \\ -1 & \textit{for hyperbolic coordinates} \end{cases}$, d_i=±1 depending upon the direction of rotation, and $e^i = \begin{cases} \tan^{-1} 2^{-i} & \textit{for circular coordinates} \\ 2^{-i} & \textit{for linear coordinates} \\ \tanh^{-1} 2^{-i} & \textit{for hyperbolic coordinates} \end{cases}$.

Hence, more elementary functions can be computed using CORDIC, such as division, multiplication, square root, trigonometric functions, inverse trigonometric functions, logarithmic functions, multiply add operation, divide add operations, and hyperbolic functions. In a nutshell, CORDIC is an algorithm capable of computing a wide range of elementary functions using simple shift and add operations.

Besides being used for computing elementary mathematical functions, it has been applied in many digital signal processing (DSP) applications, such as speech synthesis, fast Fourier transform (FFT), Discrete Fourier Transform (DCT), matrix arithmetic, and digital filtering. It has also been extensively used in robotic applications, such as inverse kinematics. As It can compute many useful functions, a number of recent applications require CORDIC as a basic processor, e.g., video compression, video conferencing [9] [10], fast cable modems, and co-processor of super computers.

Advantages of CORDIC include, but not limited to, a single algorithm capable of computing a wide range of arithmetic functions, and can perform the required computations without using a multiplier, which was very expensive at that time. Moreover it can perform many functions with comparable delay and area to that of a division algorithm. On the contrary side, the main drawback of the CORDIC algorithm is the use of full precision carry propagate adder to compute each micro-iteration causing the algorithm delay to be $O(n^2)$. In addition, it has a scale factor that may typically be applied either in a pre-processing step of the input operands or in post-processing step of the output result causing extra delay and area overhead.

Literature portrays that a lot of work has been done to improve CORDIC. A great deal of research work concentrated on speeding up the algorithm by reducing the number of iterations of CORDIC. There are different approaches to achieve this goal. Some of the major proposed solutions are summarized here.

Ahmed [11] introduced a hybrid CORDIC that uses multiplication and look-up tables. This technique is based on advancements in VLSI technology, where multiplier and storage devices are considerably less expensive than earlier ones. Two types of hybrid CORDIC were reported. In the first type, to rotate a vector, coarse rotations are performed first using additional look-up tables and a multiplier, followed by CORDIC refined rotations. This approach allows the tradeoff between execution speed and storage size. The second type of hybrid CORDIC [12] is to perform coarse rotations using CORDIC, followed by Taylor series approximation using a multiplier. Hybrid CORDIC tried to improve CORDIC by using other evaluation techniques, e.g. Taylor series expansion, and extra hardware that is used outside CORDIC.

Timmermann, et al. used a multiplier to reduce the number of iterations [13]. This method is based on the fact that only the early iterations of CORDIC contribute significantly to the accuracy of the final result. As the iteration index increases, the result accuracy due to that iteration step decreases. Hence, in this technique, ORDIC iterations are performed up to j iterations, where $j > (n+1)/2$, for n-bit accuracy. Then a multiplication or division operation is performed in the rotation or vectoring mode, respectively. This approach reduces the number of iterations needed by adding one multiplication in the rotation mode, or one division in the vectoring mode. However, both the multiplication and division operations are expensive.

One of the most effective ways to accelerate CORDIC is to use redundant number system. This causes each micro-rotation iteration step to have a constant delay irrespective of the size of input operands or the desired accuracy, causing the algorithm to have an $O(n)$ delay instead of $O(n^2)$. The first redundant CORDIC was proposed by Ercegovac and Lang [14]. To compute the Sine and Cosine the direction of micro-rotation (di) can be determined by computing an estimate of the angle using few of its most significant digits. If this estimate is positive, the direction of rotation is selected to be counter clockwise, when it is negative a clockwise rotation is performed, or no rotation when estimate is zero. As there is no rotation when the angle estimate is zero, the number of iterations is not constant and accordingly the scale factor is no longer constant in which case it has to be calculated along with the rotated vector coordinates causing additional delay and hardware overhead.

Antelo and Bruguera [15] proposed a radix 2-4 redundant CORDIC that has constant scale factor and can perform vectoring and rotation mode in hyperbolic and circular coordinates. However, it is a complex algorithm that has three kinds of special cases during the iterations and also pre-scaling of input operands x and y is required. Dawid and Meyr [16] proposed a radix 2 redundant number system CORDIC called Differential CORDIC. It transformed the original CORDIC algorithm to a redundant one. This resulted in constant scale factor, new variables, and different sign estimation method. They also derived parallel architectures for the rotation as well as the vectoring modes. This solved the variable scale factor problem of redundant CORDIC and avoids additional operations compared to other redundant CORDIC solutions.

Duprat and Muller [17] have presented a branching CORDIC algorithm using binary singed digit as redundant number system with digit set being [-1, 0, 1]. They came up with a constant scale factor without modifying the basic CORDIC rotation. In their technique they are using two parallel CORDIC architectures. In each rotation iteration they estimate the sign of remaining angle (Z_i) by inspecting δ significant digits of it, and perform a positive or negative rotation on both architectures (parallel CORDIC modules) as long as they are sure about the sign of Z_i (value of δ bits >0, or <0). When they are unsure about the sign (the value of δ signed digits = 0) they branch and perform a positive (clockwise) rotation on one module and negative (counter clockwise) rotation on the other module. Branching continues until either the sign of positive module (Z_i^+) becomes +ve or the sign of negative module (Z_i^-) becomes –ve. In either case branching is stopped and normal CORDIC rotations are continued until the next branching condition occurs. In this way they have come up with a fixed number of rotations which guarantees the constant scale factor. They have achieved faster CORDIC operation with fixed scale factor and without changing the basic CORDIC iteration but at the cost of double the area of standard CORDIC.

In [18] Tayler series expansion is used to design a scale free CORDIC. The micro rotation angles are restricted to have only single direction such that algebraic sum of these micro rotation angles forms the input angle. Then cosine and sine functions are approximated to 3rd order Tayler series expansion. However this approximation imposes the restriction on starting iteration index to be shifted up which results in very small Region of Convergence (ROC). For example, for 16 bit precision the iteration index starts with 4 which results in very low ROC i.e. only7.16^o.

A modification of this method was presented in [19]. Authors have used the domain folding technique to extend the region of convergence to entire coordinate space. Further, they have used a preprocessing unit to map the micro rotation angles to achieve a scale free CORDIC. But drawback of this technique is that it also requires a post processing unit to implement the adaptive scale factor. A new CORDIC to evaluate sine and cosine functions with variable scale factor was proposed in [20].It can reduce the number of iterations for 16-bit numbers to a maximum of six iterations and an average of 4.5 iterations. This is accomplished by scanning two bits at a time instead of checking the sign-bit only. The algorithm works with fixed point numbers for angles between ±1.

Some researchers have addressed the issue of range of convergence for CORDIC. Two ways to solve the CORDIC range of convergence limitation have been reported. First way is known as input argument reduction, and the second one is called CORDIC convergence domain expansion. In the first approach, the argument is reduced, then it is evaluated, and the final result is constructed according to the original reduction. Walther [8] has proposed a set of reduction functions for the input arguments. For example, to evaluate sine of a large argument ‘theta’ the angle is first divided by pi/2 producing a quotient Q and a remainder D with |D| < pi/2 which falls in the range of convergence. However, this method requires more chip area for VLSI implementation and the reduction time may exceed the actual evaluation time of CORDIC, in addition to control penalty and one division operation. Haviland and Tuszynski [20] suggested a pre rotation approach. If a vector falls outside the convergence range, then rotations by pi/2 and pi/4 are performed. This method results in both extra execution time and control overhead. Hahn, et al. [21] proposed an argument reduction CORDIC with ``unlimited" range of convergence to deal with floating point CORDIC. This approach uses shifters, multiplexers, carry-save adders and a ROM which makes this approach a bit complicated and expensive in terms of area.

A CORDIC-based pipelined architecture [22] to perform Givens rotations was proposed for filters. This architecture is for normalized lattice all-pass filters. By using pipeline interleaving technique, a large number of filters can be obtained. This technique fully exploits the pipeline property of the Givens rotation processor regardless of the recursive form of the infinite impulse response all-pass filters.

Hekstra and Deprettere [23] proposed a full precision floating point CORDIC to avoid the accuracy problems. As the main drawback in floating point computation of the standard CORDIC algorithm occurs in the inherent fixed point resolution of the angle. When calculating angles close to or smaller than the angle resolution, the inaccuracy becomes unacceptable. In this approach, angles are represented as a combination of exponent, micro rotation bits and two bits to indicate pre rotations over pi/2 and pi radians to achieve higher accuracy.

A redundant and on-line CORDIC was proposed in [14]. This new CORDIC found its applications in matrix triangularization and Singular Value Decomposition (SVD). The major modifications of this CORDIC can be summarized in the following. First, redundant addition is used for calculating the rotation angle $\tan^{-1}(y/x)$. This approach is faster than using carry propagate adders, but results in a variable scale factor. Second, the angles are transmitted in decomposed forms. This eliminates the recurrences of angles in both CORDIC modules and reduces the communication bandwidth. Finally, on-line addition is used in the implementation of the rotation modules.

A multi-dimensional CORDIC algorithm was proposed by Hsiao and Delosme [24] called Householder CORDIC. By expressing matrix computations in terms of higher dimensional rotations, they can be implemented using the Householder CORDIC. For complex large matrix computations, the maximum throughputs of parallel arrays built out of CORDIC units that process two real numbers at a time are very low compared to the throughputs achievable for real data sets of equal size. To bring the throughputs closer to real data throughputs, further parallelism must be explored. The rotation concept of Householder CORDIC was extended to vectors in spaces with more than two dimensions [25]. This method was employed to speed-up the computation of the singular value decomposition of complex matrices.

2.2 Multipliers

A binary multiplier is an electronic circuit used in digital electronics, such as computer, to multiply two binary numbers. It is built using binary adders. A variety of computer arithmetic techniques can be used to implement a digital multiplier [26]. Shift-and-add multiplication is the basic technique of multiplication and is similar to the multiplication performed by paper and pencil. This method adds the multiplicand ‘Y’ to itself ‘X’ times, where ‘X’ denotes the multiplier. The algorithm proceeds by taking the digits (bits) of the multiplier one at a time from right to left or left to right, multiplying the multiplicand by a single digit of the multiplier and placing the intermediate product in the appropriate positions to the left or right of the earlier results for addition. Several methods have been proposed to speed up the multiplication process e.g. high radix multipliers, Array multipliers, etc. Some of these fast multipliers are reviewed below.

Array multiplier are well known due to its regular structure. The basic algorithm in array multipliers is add and shift. Partial products are generated by multiplying the multiplicand with one of the multiplier bits and then are shifted according to their bit order before addition. Intermediate addition steps are performed by a carry-save addition method and the final product is obtained by using a fast adder (like carry look ahead adder, etc.). Number of partial products to be added in an array multipliers is equal to the number of bits in multiplier operand. Now as multiplier’s operands may be negative or positive so 2’s complement number system is used to represent operands. At each carry-save stage all numbers to be added should be of the same size. Therefore some sign-bit extension is needed at each stage of carry-save adders. This sign bit extension results in a higher capacitive load (fan out) of the sign bit signals compared to the load of other signals and accordingly results in slower speed of the overall circuit [27]. An algorithm that eliminates the need for the common sign bit extension in addition is implemented in [28] and [29]. This not only leads to a drop-off in capacitive load of the intermediate sum/carry sign-bit signals (reduced delay) but also results in reduction of the circuit area.

Another elegant method of multiplication is Booth algorithm which gives a uniform procedure for multiplication of sign and unsigned operands. In this algorithm partial products are generated by only shifting and there is no need of addition to form the partial products in binary and in radix 4 booth algorithm. Modified Booth recoding algorithm [30] is one of the most popular techniques to reduce the number of partial products to be added while multiplying two numbers. In booth encoding, multiplier digit set is encoded into a balanced digit-set (-r/2 – r/2) which results in reduction of number of pre-computations from (r/2-1) to (r/4-1). It means there are no pre-computations in case of radix-4 multiplier. This is a great saving in terms of silicon area and also speed, as number of stages to be added is reduced to half compared to normal add and shift multiplication.

A very important iterative realization of parallel multiplier was introduced by Wallace [31] with an aim to improve the speed of a parallel multiplier. This advantage becomes more distinct for multipliers with larger operands. In Wallace tree algorithm, all bits of partial products are added together in each column by a set of counters in parallel without any kind of carry propagation. Another set of counters then reduces this new matrix and this process goes on until a two-row matrix is generated, here a 3:2 compressor is used. Then, a fast adder is used at the end to produce the final result. The advantage of Wallace tree is fast speed because the addition of partial products is now O(IogN) where N is the size of input operands.

Parallel multipliers occupy more silicon area and consume more power so it is not wise to use them in the applications where area and power are strictly restricted and speed is not a critical issue. In such situations, twin pipe serial parallel [32] multiplier is the better choice. In this multiplier odd and even indexed data bits are processed in different circuits and on different clock phase. Hence throughput is doubled due to two bits processing in one clock cycle. In contrast to parallel multipliers where the delay is mainly caused by partial product stages, piped serial parallel multiplier's delay is due to its internal loops in each multiplication stage. So, the main problem in this kind of multiplier is to reduce internal loop delay which is the only bottleneck in throughput.

Another fast multiplier is based on column compression multiplier called Dadda [33]. This multiplier, like Wallace [31], consists of three stages. In first stage partial products are produced while in second stage these partial products are reduced to only two and in the final stage a carry propagate adder is used to get the final output. A good comparison of different multipliers based on various parameters like area, delay, power, area-delay product, and power-delay product, etc. can be found in [27], [34], and [35].

2.3 Dividers

Division is the most expensive operation among the all basic arithmetic operations. Thus designing a fast divider is very important in high speed computing. There are mainly two categories of division techniques, first one is the digit recurrence and the second one is based on numerical methods like Newton-Raphson [36], [37] and Goldschmidt [38]. Digit recurrence method is slower as compared to numerical methods' technique as it produces only single quotient digit in one iteration of algorithm. Techniques like restoring division, non-restoring, and SRT [39] division fall into the digit recurrence category of the division. An improved restoring division technique is presented in [40]. Ercegovac and Lang [41] proposed an improved algorithm of SRT division using quotient digit prediction method while online SRT division methods are investigated in [42]–[44].

2.4 Square rooters

There are two main families of algorithms for square rooting namely digit recurrence and the convergence square rooting methods. Digit recurrence produces one digit (one bit for binary) per iteration of the algorithm. Early microprocessors which didn't have hardware multipliers used this kind of square rooting methods [45]. Furthermore, this approach was the apparent choice for earlier FPGAs which didn't have the built-in multipliers, thus most FPGA implementations in vendor tools and literature realized this approach [46], [47]. The second category of square root techniques was introduced as soon as microprocessors included hardware multipliers, this family uses multiplication and addition operations to compute square root with quadratic convergence. This method has been primarily derived from Newton-Raphson iterations and was firstly used in AMD processors[48]. Some other variations consist of piecewise polynomial approximation [49], [50] and array square-rooters [51].

2.5 Tangent inverse function evaluators

Tangent inverse or arctangent(x) or $\tan^{-1}$ (x) is a trigonometric function which is defined for all real numbers. There are two types of arctangent functions based on the number of arguments they evaluate. First type accepts either single argument or two arguments as input (as in, θ = arctan(x) or θ=arctan(y/x)) and produces the output in the range of $[-\pi/2, \pi/2]$. Whereas, second type is a variation in the arctangent function and is called a four-quadrant arctangent function. It accepts two arguments as input (as in θ=atan2(y/x)) and provides output in the interval of $[-\pi, \pi]$. Although it is now common to almost all fields of science and engineering, it was first introduced in different programming languages of computer like C/C++[52], MATLAB [53], Mathematica [54], Java [55], etc. The atan2 function takes into account the signs of both vector components.

There exist a lot of approaches to implement arctangent functions. These include Taylor series expansion, iterative algorithms such as CORDIC, Look–up table based approaches, and polynomial and rational function approximations. Taylor series expansion is a direct way of computing arctangent functions but it converges slowly when the argument is near to one which makes it inefficient technique. CORDIC uses only add and shift operations and can be successfully used to compute trigonometric functions [56], [57]. This method is very attractive where the area is major concern. Look-up table based approach is very fast and straightforward way to compute inverse trigonometric functions. But for n-bit operands, it requires a memory size of $n \times 2^n$ making it not suitable for the designs with $n \geq 20$ bits input operands [58]–[60].

Polynomial and rational function approximations are also other elegant ways of computing trigonometric functions. But these techniques use multiple multiply-add units which makes them unsuitable for area sensitive applications [61].

CHAPTER 3

PROPOSED HARDWARE DESCRIPTION

In the inertial navigation system, measurements are taken and processed at a regular sampling period Ts based on which various control signals are generated for guidance of the vehicle. Due to periodic nature of the input and the small sampling period, hardware implementation can achieve high throughput using a pipeline architecture [62]. Therefore, the proposed hardware is a combination of iterative and pipeline approaches, where individual components of the pipeline stages have been implemented using iterative technique. The proposed hardware is described in this chapter.

3.1 Design assumptions

Following assumptions were made in the hardware design of the IINS system.

1. There exists a host system that translates application commands and sensors' measurements into the proper inputs of IINS processor. Any standard interface can be used between the host system and the IINS processor.
2. A start (START) signal initiates the computations.
3. The input to output latency varies depending upon the input vector length. Thus a job completion signal BUSY is provided to indicate the availability of valid results.

4. The combination of START and BUSY signals makes the IINS processor independent of any particular input or clock speed, which makes it easily programmable and interface able to many platforms.
5. Fixed point arithmetic with 32 bit word length for 16 bit input operands has been used in the design of this system.

3.2 Design inputs and outputs

IINS hardware takes sampling period Ts, $Ts/4\mu$, and measurements of 18 accelerometers as inputs. These are n bit fixed point inputs with 2 bits before the decimal point and n-2 bits after it. Both inputs and outputs are represented in 2's complement form. After processing these inputs the IINS hardware provides angular velocities $(\omega_1, \omega_2, \omega_3)$, quaternion parameters (q1, q2, q3, and q4), DCM elements, and attitudes $(\theta_1, \theta_2, \ and\ \theta_3)$ of the vehicle as output. Angular velocities are the 2n-2 bit long values with 1 sign bit, 1 integer bit and 2n-4 fractional bits, the quaternion parameters and the DCM elements are 2n bits with 2n-2 fractional bits, 1 sign bit and 1 integral bit before the decimal point, whereas attitudes are n bit quantities with 9 integral bits (including 1 sign bit) before the decimal point and n-9 fractional bits. Figure 3.1 shows the input and output ports of the designed IINS processor.

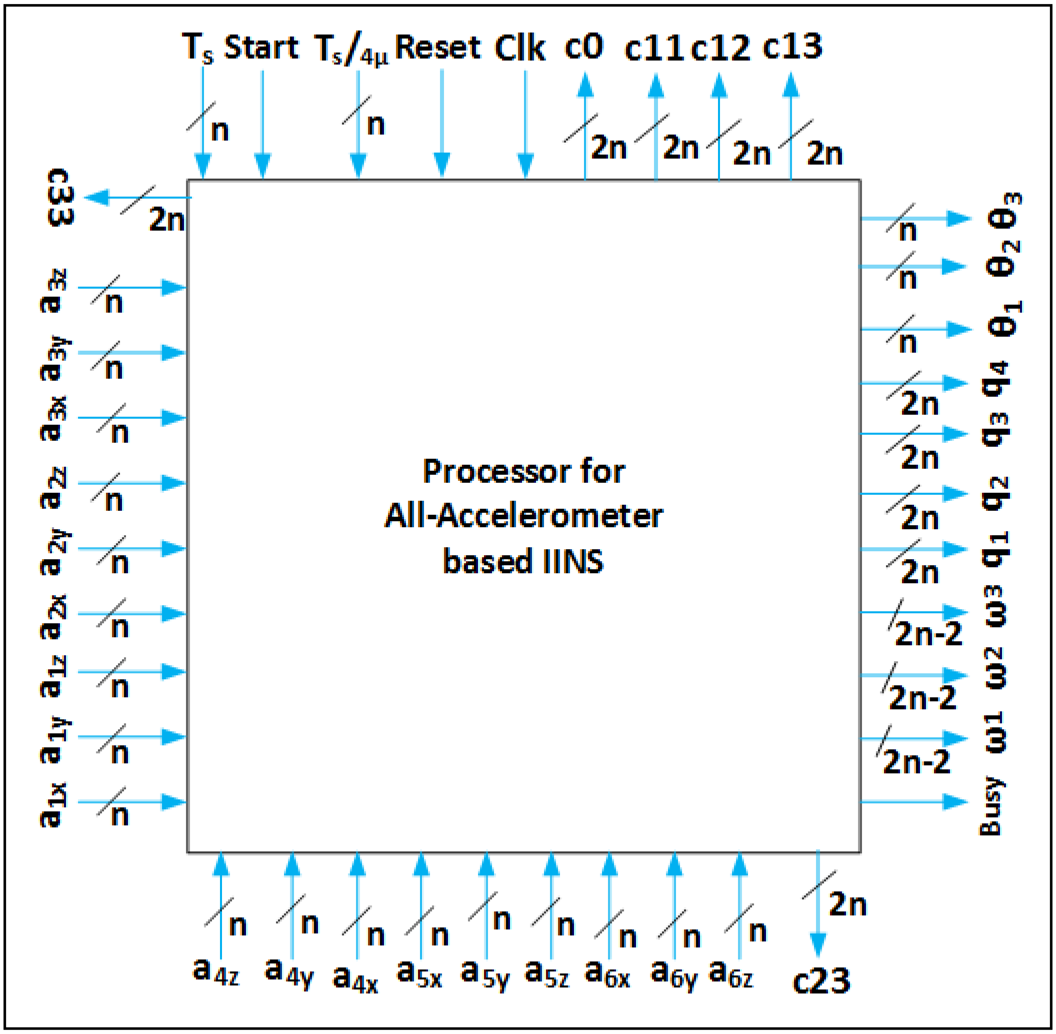


Figure 3.1: IINS processor input and output ports

3.3 VHDL design methodology

The task is to follow a systematic approach to develop an algorithm-specific hardware architecture. This design strategy is broken down into making decisions in the algorithm and in the architecture space. Following figure shows the main decisions that have been considered for both spaces [63].

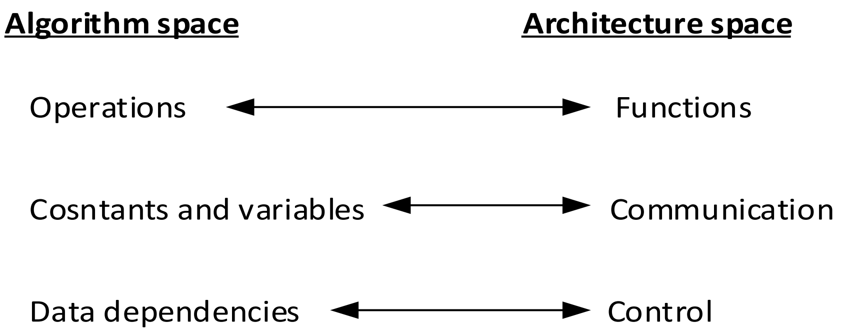
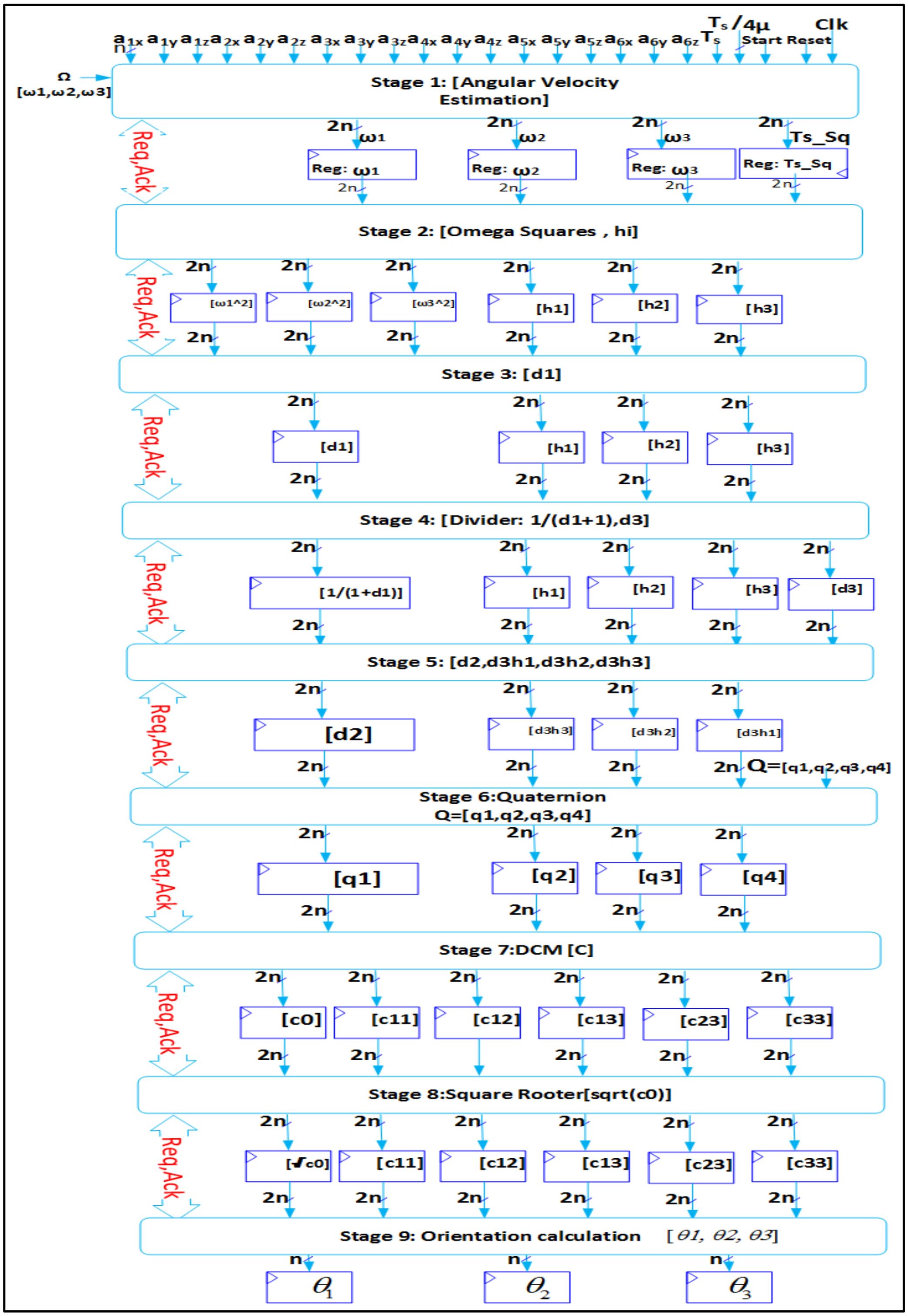


Figure 3.2: Mapping of design issues in algorithm and architecture space.

3.4 Hardware design break down

Figure 3.3 depicts the proposed hardware of the IINS system consisting of 9 pipeline stages. Each pipeline stage is responsible to execute some specific task in order to implement the equations involved in the IINS. Now, every stage of the hardware is described briefly.



a1x a1y a1z a2x a2y a2z a3x a3y a3z a4x a4y a4z a5x a5y a5z a6x a6y a6z Ts Ts/4μ Start Reset Clk
n
Ω [ω1,ω2,ω3]
Stage 1: [Angular Velocity Estimation]
Reg: ω1
Reg: ω2
Reg: ω3
Reg: Ts_Sq
Req,Ack
Stage 2: [Omega Squares , hi]
[ω1^2] [ω2^2] [ω3^2] [h1] [h2] [h3]
Stage 3: [d1]
[d1] [h1] [h2] [h3]
Stage 4: [Divider: 1/(d1+1),d3]
[1/(1+d1)] [h1] [h2] [h3] [d3]
Stage 5: [d2,d3h1,d3h2,d3h3]
[d2] [d3h3] [d3h2] [d3h1]
Q=[q1,q2,q3,q4]
Stage 6:Quaternion Q=[q1,q2,q3,q4]
[q1] [q2] [q3] [q4]
Stage 7:DCM [C]
[c0] [c11] [c12] [c13] [c23] [c33]
Stage 8:Square Rooter[sqrt(c0)]
[√c0] [c11] [c12] [c13] [c23] [c33]
Stage 9: Orientation calculation [θ1, θ2, θ3]
2n
n
θ1 θ2 θ3

Figure 3.3: Different stages of the IINS hardware.

3.4.1 Request-Acknowledge (*Req-Ack*) Protocol

We have implemented a request-acknowledge (*Req-Ack*) protocol between every two stages of the pipeline. This protocol is responsible for flow control, so that faster stages of the pipeline should not overwhelm the slower ones. In this protocol, sender stage may send a *Req* signal while receiver stage has an *Ack* signal to indicate start of computations for the requested job. When there is no data to be sent/received (i.e. quiet state) both *Req* and *Ack* signals are zero. When sender stage has finished its computations, it asserts the *Req* signal at rising edge of the clock and waits for a response from the receiver. When receiver detects a rise in the *Req* signal, if it has completed its previous job (*Busy* signal is low), it asserts the *Ack* signal and starts processing the requested job. Sender stage de-asserts its *Req* signal when it detects a rise in *Ack* signal of the receiver stage. The receiver stage, in turn, also lowers its *Ack* signal when it detects a low *Req* signal from the sender. This 4-phase Request-Acknowledge protocol is shown in the figure below.

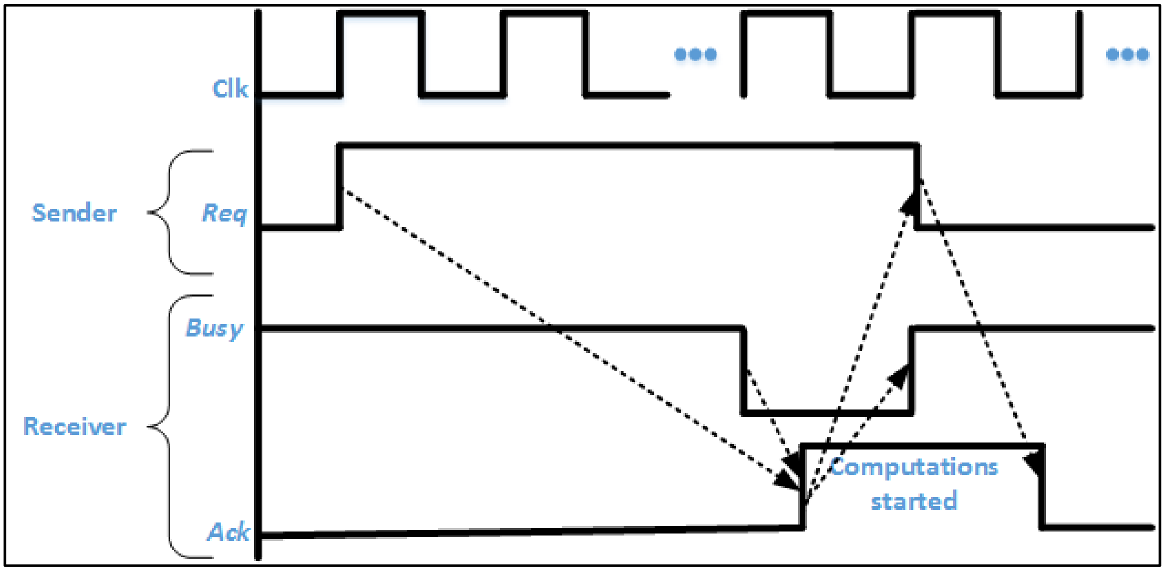


Figure 3.4: Timing diagram of *Req-Ack* protocol

3.4.2 Stage 1: Angular Velocity Estimation

The first stage of the pipeline in the IINS hardware is responsible for estimation of the angular velocities (equation(1.11)) given angular accelerometers' measurements, Ts, and µ as inputs. This stage uses three multipliers and one squarer to compute $\omega_1, \omega_2, \omega_3, and\ Ts^2$ as shown in Figure 3.5.

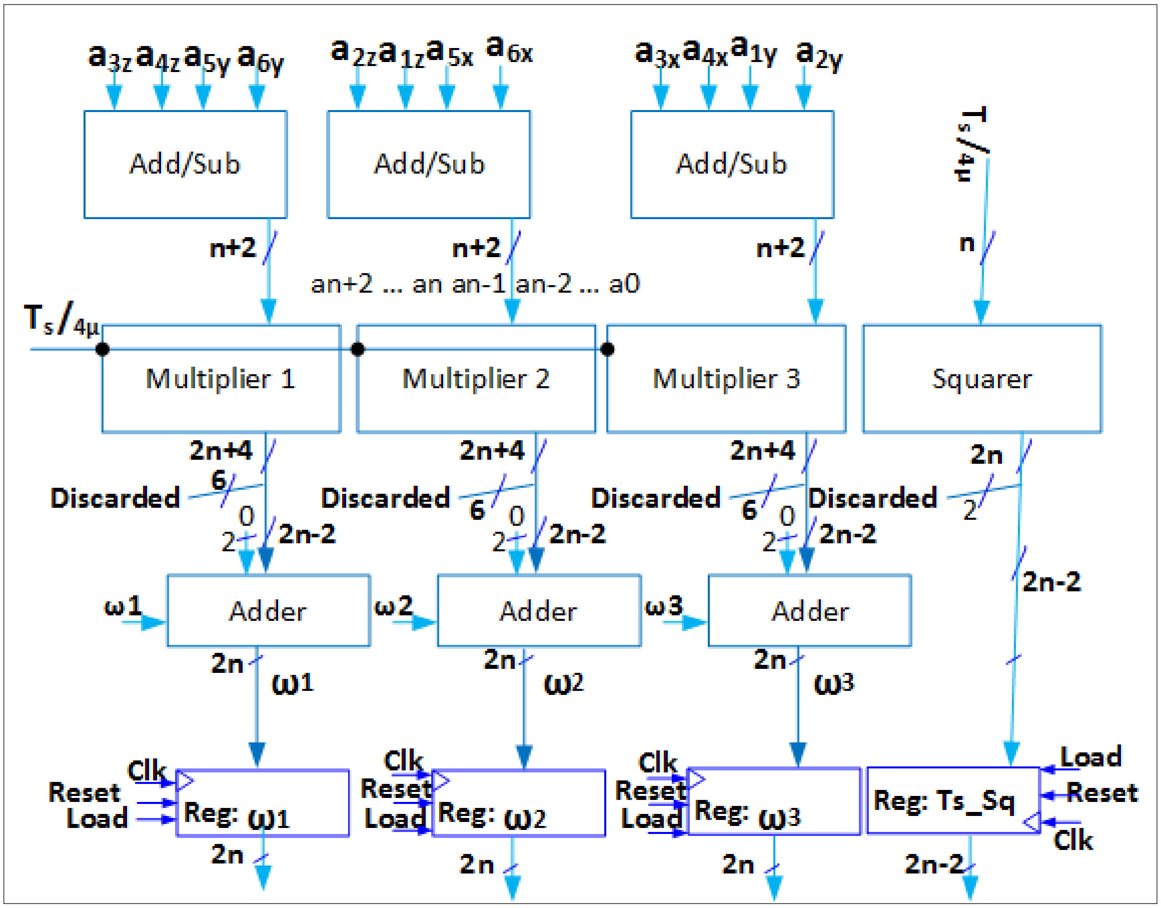


Figure 3.5: Data path of the angular velocity estimation module

3.4.3 Stage 2 and 3: Computation of omega squares, h_i's and d1

Second stage computes squares of angular velocities ($\omega_1^2, \omega_2^2, \omega_3^2$), h1, h2, h3, and d1 variables, where h_i (h1, h2, h3) is the product of i^{th} component of angular velocity with the sampling period Ts (equation(1.16)). Note that we are calculating h_i components prior to their requirement in order to eliminate extra delay for computation of quaternions in the quaternion stage. Stage 2 uses 3 squarers, and three multipliers to compute $\omega_1^2, \omega_2^2, \omega_3^2$, h1,h2, and h3. After completing its computations, stage 2 passes its output to stage 3 of the pipeline using *Req-Ack* protocol. Stage 3 comprises of a multiplier and a couple of adders to compute d1 (equation(1.17)). It passes d1, and h_i components to the next stage after completion of its computations.

3.4.4 Stage 4 and 5: Computation of d2, d3 and d3×hi's signals

Computation of intermediate signals d2 and d3 requires a divider (equation(1.17)). The equation(1.17), in its essence, suggests that two dividers should be used to compute d2 and d3, but we have modified the equation in such a way that only 1 multiplier and 1 divider are required which results in reduced area. A single divider has been used to compute the inverse of (1+d1) and then this quantity is shifted 1 bit right to get d3 while d2 has been calculated by multiplying the result of divider with (1-d1).

To generate 1+d1(d1 is a 2n bit fixed point number with 2 bits before decimal point and 2n-2 bits after the decimal point), instead of using a full CPA, only most significant two bits of the d1 are replaced with "01" to avoid the 2n bit full CP adder delay. This tweak was possible because most significant two bits of d1 are always zero. Then an SRT divider as shown in Figure 3.10 is used to compute the inverse of 1+d1. Stage 4 produces inverse of (1+d1) and d3 using an SRT-divider module while the stage 5 generates d2 (equation(1.17)), d3xh1, d3xh2, and d3xh3 using 4 multipliers (equation(1.16)).

Furthermore, in order to reduce the delay of the Quaternion stage we are computing d3×h1, d3×h2, and d3×h3 in parallel with the calculation of the d2 (equation(1.17)) in 5th stage.

3.4.5 Stage 6: Quaternion (q1, q2, q3, q4) Computation

Once we have calculated d2 we are ready to generate quaternion parameters (equation(1.18)). If we had implemented the equation (1.18) as it is, a total of 28 multipliers and three successive multiplications would have been used. But identifying some common expressions ($d3 \times h1, d3 \times h2, d3 \times h3$) in the quaternion equations, we were able to reduce the number of multipliers to 16, but still three successive multiplications were required. Then it was noted that some variables can be calculated prior to this quaternion stage as soon as their input data is available (e.g. h1, h2, h3 could be calculated in stage2 while d3xh1, d3xh2, and d3xh3 could be calculated in stage 5). These variables' values can be brought forward to the quaternion stage using some extra registers. In this way, we reduced the delay of quaternion stage from three successive multiplications to that of a single multiplication.

Moreover, as we wanted to maintain minimum possible area of the hardware design, we decided to use only 8 multipliers instead of 16 in this stage and repeated these 8 multiplications twice to generate the quaternion variables using a controller and some multiplexers. This stage accepts previous values of quaternion parameters (q1(t-1), q2(t-1), q3(t-1), and q4(t-1)), d2, d3xh1, d3xh2, d3xh3, and d2 as inputs and produces new values of quaternion parameters (equation(1.18)) as shown in Figure 3.6.

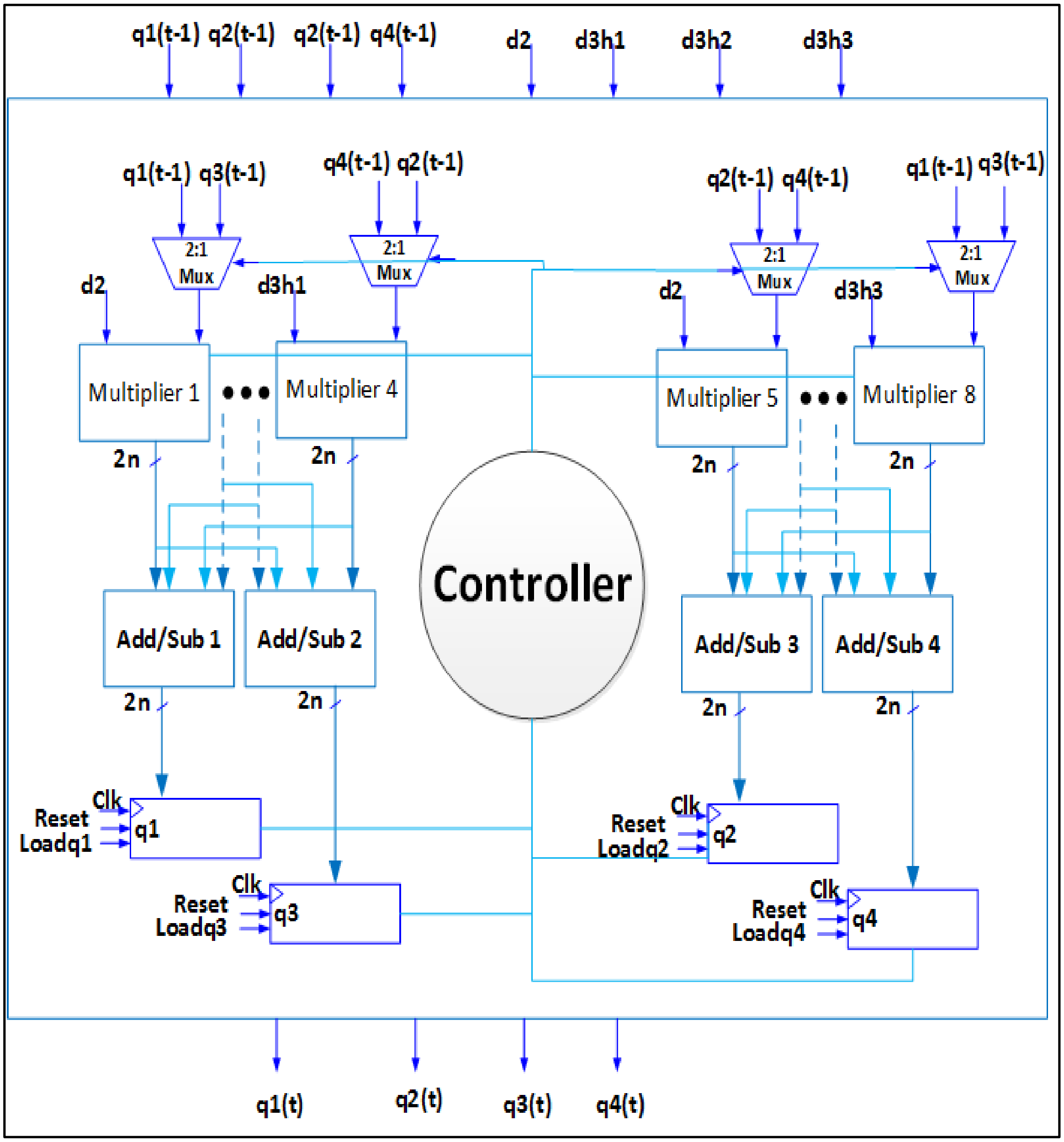


Figure 3.6: Data path of the quaternion stage

3.4.6 Stage 7: Directional Cosine Matrix

The computation of Directional Cosine Matrix (DCM) involves 15 multipliers and two successive multiplications, if the equation (1.19) were implemented as it is. We decided to use only 8 multipliers (actually 4 squarers and 4 multipliers) along with a finite state machine controller and 3 multiplexers to reduce the area while having the same delay for this stage.

The DCM stage accepts quaternion parameters q1, q2, q3, and q4 as input and produces directional cosine matrix components c0, c11, c12, c13, c23, and c33 as output. Figure 3.7 shows the data path of the DCM stage. Four squarers are used in this stage to produce $q_1^2, q_2^2, q_3^2, q_4^2$, then their outputs are used to compute c11 and c33 (1.19) using two four-input add/sub modules. Similarly, 4 multipliers are responsible to produce other components of the directional cosine matrix with two iterations of these multipliers. During the first iteration, multiplier 1 to multiplier 4 (multiplier 2 and 3 are shown as dots) produce products q1×q2, q1×q3, q3×q4, and q1×q4 respectively. Whereas, during the second iteration multiplier 1 and 4 produce products q1q2q3q4 and q2q4 respectively. Then by adding or subtracting these products we compute c0, c11, c12, and c23 (equation(1.19)) as shown in the Figure 3.7.

All inputs of the multipliers are controlled using multiplexers and appropriate control signals generated by the controller. Similarly, all registers are loaded with their appropriate values using the load signals coming from the controller.

3.4.7 Stage 8: Square Rooter

This stage accepts c0 as input and produces $\sqrt{c0}$ as output. It uses one CORDIC processor and one multiplier in succession to produce the required square root. CORDIC generates the square root while multiplier is used to perform some post-processing required by the CORDIC generated square root result.

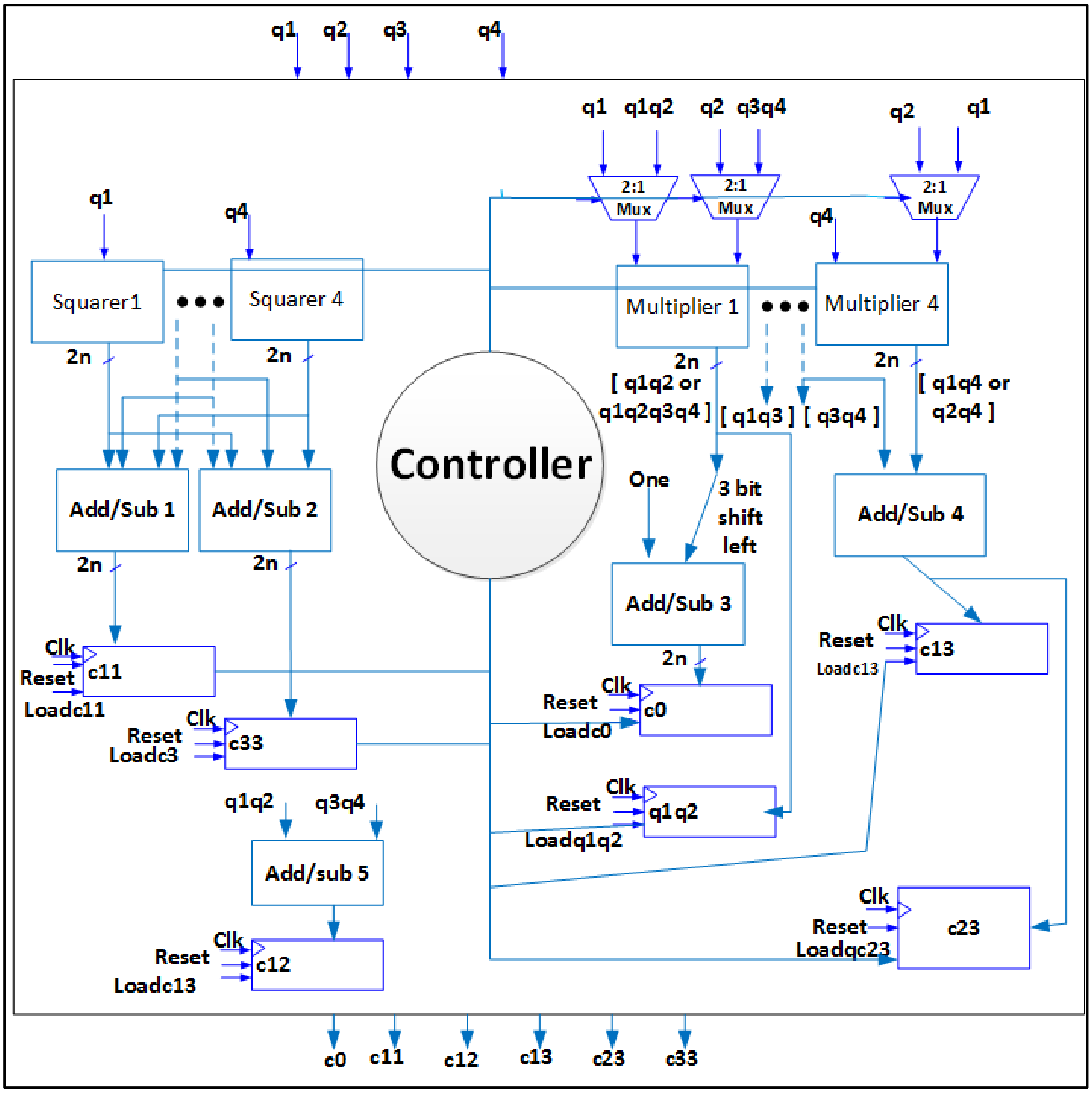


Figure 3.7: Data path of the DCM stage

3.4.8 Stage 9: Orientation of the aerial vehicle

Attitude or orientation of the aerial vehicle is computed using quaternion to Euler angles ($\theta_1, \theta_2, \theta_3$) conversion using equations (1.20) to (1.22). Normal atan function has a range of (-$\frac{\pi}{2}$ to $\frac{\pi}{2}$). But it is clear from equation(1.20), these atan functions have a range of (-π to π). These kind of atan functions are called 4-quadrant atan functions.

It is obvious from equations (1.21) and (1.22) that conversion from quaternion to Euler angles requires a magnitude comparator, a divider and an atan function calculator. This definition of Euler angle conversion is very costly both in terms of area and delay. But luckily there exist one better, and hardware friendly definition for calculating this kind of 4-quadrant atan functions [64]. In the literature, these functions are referred to atan2 functions defined as follows:

$$\operatorname{atan2}\ (y,x)=\begin{cases}\tan^{-1}(y/x) & if\ \ x>0\\ \tan^{-1}(y/x)+\pi & if\ \ y\geq 0, x<0\\ \tan^{-1}(y/x)-\pi & if\ \ y<0, x<0\\ +\dfrac{\pi}{2} & if\ \ y>0, x=0\\ -\dfrac{\pi}{2} & if\ \ y<0, x=0\\ undefined & if\ \ y=0, x=0\end{cases}\tag{3.1}$$

Using this definition of Euler angle conversion $\theta_1, \theta_2, and\ \theta_3$ are calculated as follows

$$\begin{aligned}\theta_1 &= a\tan 2\left(-C23, C33\right); && where: -180\leq\theta_1\leq 180;\\ \theta_2 &= a\tan 2\left(C13, \sqrt{C_o}\right); && where: -90\leq\theta_2\leq 90;\\ \theta_3 &= a\tan 2\left(C12, C11\right); && where: -180\leq\theta_3\leq 180;\end{aligned}\tag{3.2}$$

Where, atan2 is defined as above. This definition is very efficient with hardware point of view because it requires only an arctangent calculator (e.g. CORDIC) and a comparison with zero. This stage accepts DCM as input and uses this definition of Euler angle conversion ((3.1) -(3.2)) to calculate the attitude of the aerial vehicle. It uses 3 CORDIC atan2 modules (Figure 3.12), and 3 multipliers to convert the final result from radian to degrees.

3.4.9 Individual components used in different stages of pipeline

In this section, hardware design of individual components used in different stages of the pipeline implementation of the IINS are discussed briefly along with their architectural details. As mentioned earlier every individual component has been implemented using iterative approach to maintain possible minimum area of the ASIC design.

Multiplier and squarer modules:

The multipliers and squarers which have been used are modified Booth radix-4 multipliers to minimize the required number of iterations to half of input operand bits to produce the product result. Figure 3.9 shows the data path of the radix-4 modified Booth multiplier. This multiplier accepts two input operands, multiplier (B) and multiplicand (A), and produces the multiplication result as P=A*B in (p+1)/2 clock cycles where p is the number of bits in input operand B.

In radix-4 Booth multiplication, multiplier (B) digits are encoded into a balanced digit set [-2, 2] so that no pre-computations of multiplicand (A) are required for generation of its multiples. The multiplier operand bits are scanned from LSB to MSB using a block of 3 bits at a time such that every new block overlaps one bit from the previous block using 1-bit look-behind method. This encoding is shown in Table 3.1.

Table 3.1 : Radix-4 modified Booth multiplier digit encoding

x_{i+1}	x_i	x_{i-1}	y_{i+1} y_i	Encoded Digit Value
0	0	0	0 0	0
0	0	1	0 1	1
0	1	0	0 1	1
0	1	1	1 0	2
1	0	0	-1 0	-2
1	0	1	0 -1	-1
1	1	0	0 -1	-1
1	1	1	0 0	0

As shown in the above table x_{i+1} x_i x_{i-1} are the current 3 bits to be encoded where x_{i-1} is the overlapped bit from previous block. Similarly, y_{i+1} y_i are the encoded digits whose value is shown in the right most column of the table.

It is clear from the data path that Booth multiplier has 4 registers (RA, RB, Count, and RP) with parallel load capability to store A (multiplicand) B (multiplier), Counter value (p+1)/2, and product result (P=A*B) respectively. In Figure 3.9, n and p represent the number of bits in the multiplier (B) and the multiplicand (A) operands respectively.

The state diagram of the controller for designed modified Booth is shown in Figure 3.8. It has 2 states (S0, S1). The initial state is reset state (S0). The controller stays in this state as long as the signal ‘Start’ becomes ‘1’. Then controller moves to S1 state generating load signals for registers RA , RB, count (count register is loaded with (p+1)/2 which is the number of digits in the multiplier operand) and a clear signal for register RP.

The *fsm* remains in the second state (S1) doing nothing, as long as the signal ‘Start’ is high. Once the start signal is reset to low, computations of the product start on the next active clock edge. In every clock cycle a multiple of the register RA (0,1RA, 2RA or their complemented values) is generated depending upon the Booth encoded value b_i of multiplier register’s least significant 2 bits and the look-behind bit. It is added/subtracted to/from most significant n bits of the product register RP and the result is saved to RP register shifting it to right by 2 bits.

The counter register is decremented by 1 and the multiplier register (B) is shifted right 2 bits in every clock cycle. All shifts are implemented using wired shifts. This process is repeated until the counter becomes 0. When the counter becomes 0, computations stop and the fsm moves back to state S0 issuing a ‘Done’ signal. Squarers are also implemented in the same way whit the only difference that in squarer, the multiplier (B) and the multiplicand (A) operands are same.

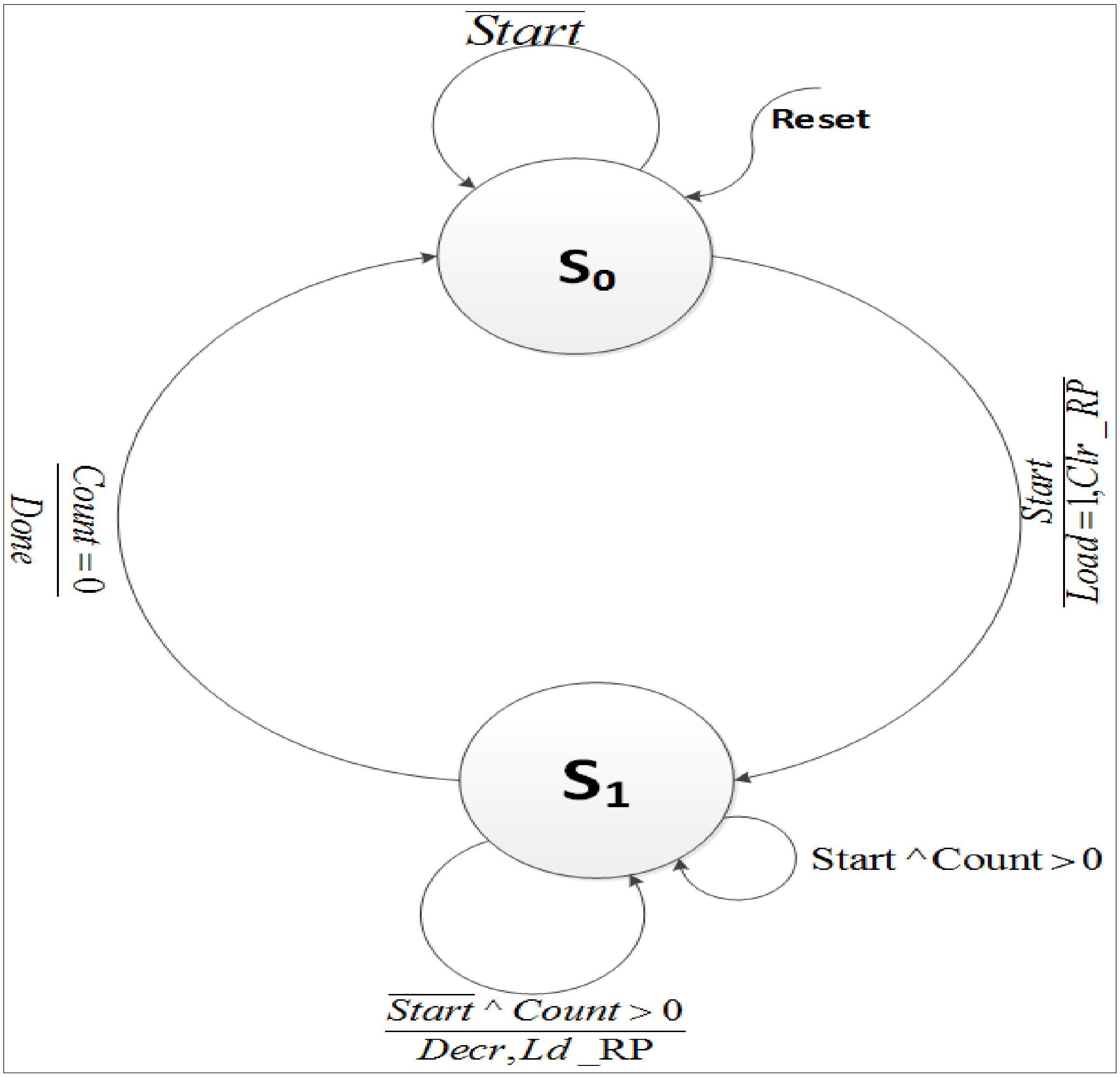

$\overline{Start}$
Reset
S_0
$\frac{Start}{Load=1, Clr\_RP}$
$\frac{Count=0}{Done}$
S_1
Start ^Count > 0
$\frac{\overline{Start} \wedge Count > 0}{Decr, Ld\_RP}$

Figure 3.8: FSM controller of the radix-4 Booth multiplier

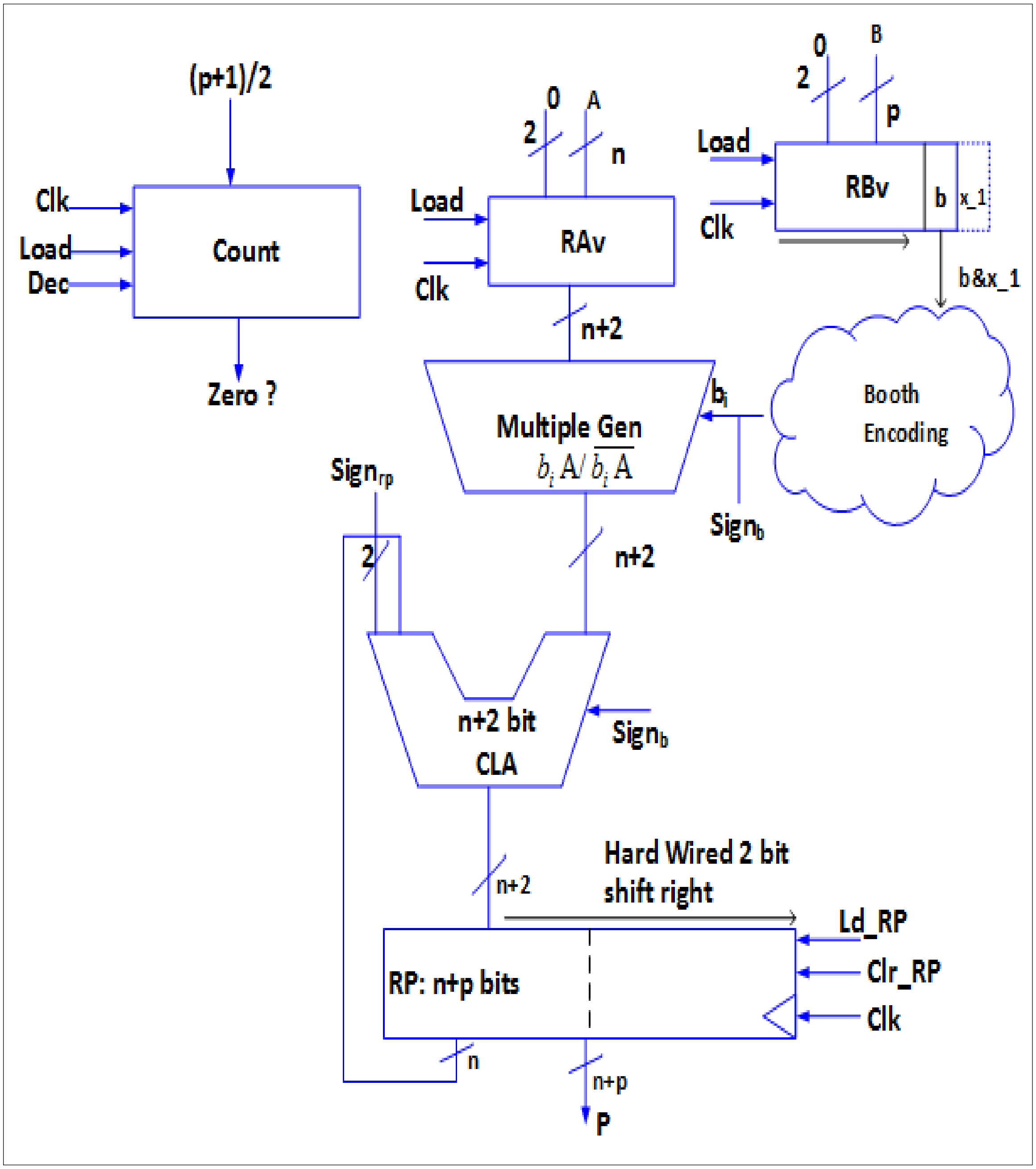


Figure 3.9: Data path of radix-4 Booth multiplier

Divider module:

An SRT binary divider was modelled to perform division operations required in (equation(1.17)) stage 4 of the pipeline. Figure 3.10 shows the data path of the designed divider. It is obvious from the data path shown below that divider accepts two input operands X (an n+1 bit 2's complement numerator), and Y(an n-bit normalized denominator) and produces a P+1 bit quotient result Q (a 2's complement fraction of the form q0 . q1 q2 … qp).

The divider data path has 4 registers, 3 multiplexers, a quotient digit selection function and an add/subtract module. The remainder register 'R' is an n+1 bit register which stores residual remainder. Counter register 'Count' is a mod P counter to keep track of number of iterations. Two quotient registers Qi and QMi are used for on-the-fly conversion of the Binary Signed Digit (BSD) quotient result. The divider accepts two input operands X, and Y and produces output (Q=X/Y) in P clocks where P is the required number of fractional bits in the division result.

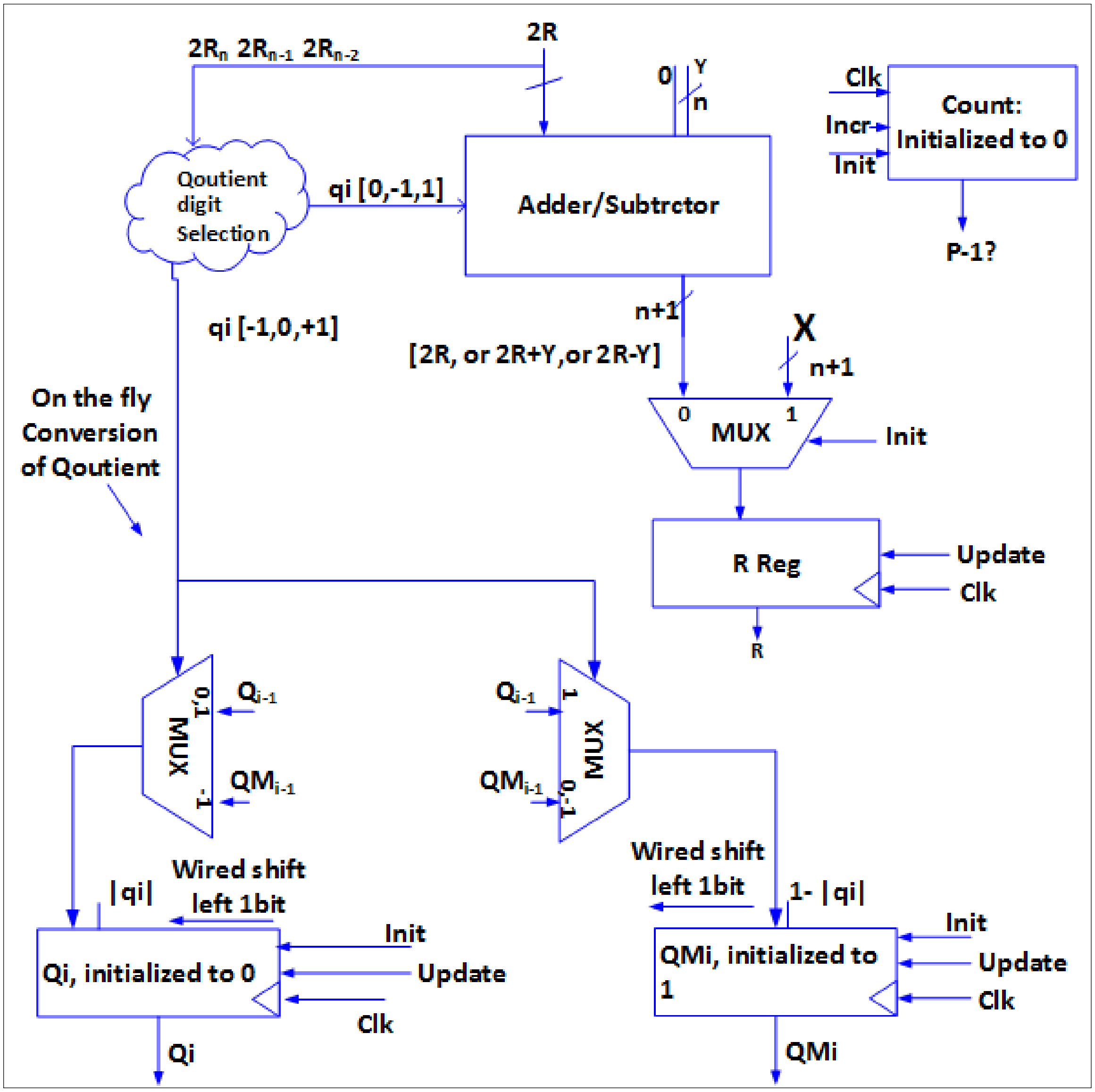


Figure 3.10: Data path of the SRT binary divider

The Counter register is initialized to zero and incremented by 1 every clock cycle until it reaches to P-1. When the value of counter reaches to P-1, computations stop and the final result is taken from the Qi register. Note that the number of required fractional bits ‘P’ in division result (Quotient) is taken as an input to the divider.

As shown in Figure 3.10 quotient digit selection function takes 3 most significant bits of the 2R and produces the quotient digit q_i. The quotient digit is selected as 1 if $2R \geq 0.5$, -1 if $2R<-0.5$ else it is 0. Please note that only most significant 3 bits of the 2R are used for quotient digit selection and one digit is produced every clock cycle.

The remainder register R is initialized with numerator operand ‘X’ and is updated either by 2R (when $q_i=0$), 2R+Y (when $q_i=-1$), or 2R-Y (when $q_i=+1$) depending upon the output of the quotient digit selection function in every clock cycle.

Selected quotient digit q_i is produced in the binary signed digit [-1, 0, 1] form. So, it must be converted to normal binary digit [0, 1] and the quotient registers should be updated accordingly. For this purpose on-the-fly [65] conversion was used as shown in the figure above by maintaining two quotient registers Qi and QMi (each one is (P+1)-bit register). Qi register is initialized to 0 while QMi is initialized to 1 and then in every iteration they are updated as follows: Q_{i-1} is shifted left by 1 bit and ‘1’ is inserted to its least significant position to update Qi while QMi gets Q_{i-1} when current quotient digit q_i is 1. When q_i is 0, Qi gets Q_{i-1} while QMi gets its previous value shifted 1 bit to left with shifted in bit being 1 (The least significant bit of QMi is 1). Similarly, the QMi register gets previous value of QMi while Qi gets the previous value of QMi shifted 1 bit left and the shifted in bit is 1 when quotient digit q_i is -1.

The above mentioned process keeps repeating until the counter register reaches P-1.Control signals like Init (short for initialize), Inc (short for increment), and update are produced using a simple fsm controller (not shown) during the whole division process.

Square rooter module:

Square rooter has been implemented using a CORDIC module and a multiplier to produce square root of c0 required in 8th stage of the pipeline. The CORDIC processor is used in vectoring mode with hyperbolic coordinates to calculate square root of any number. Figure 3.11 shows the data path of square rooter developed using CORDIC processor. As shown in the figure below CORDIC-based square rooter uses 2 registers (X and Y), one counter, 4 adders, 2 shifters, and a couple of multiplexers along with a multiplier for post-processing of the result. The register X is initialized with Xin +1/4, the register Y with Xin-1/4 while counter register is initialized to 1.

The counter is designed in such a way that its count sequence follows as 1, 2, 3, 4, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 13, 14, 15, … (values i=4,13,… i,3×i+1 are repeated). Counter is incremented every clock cycle following the mentioned sequence and shifters provide i-bits arithmetically shifted right values of X and Y registers where i is the value of the counter in current iteration.

Y and X registers are updated with their previous values added/subtracted to/from shifted values of X and Y registers (equation (2.3)) respectively depending upon the value of d_i (d_i=Not(sign(Y))) as shown in the below data path.

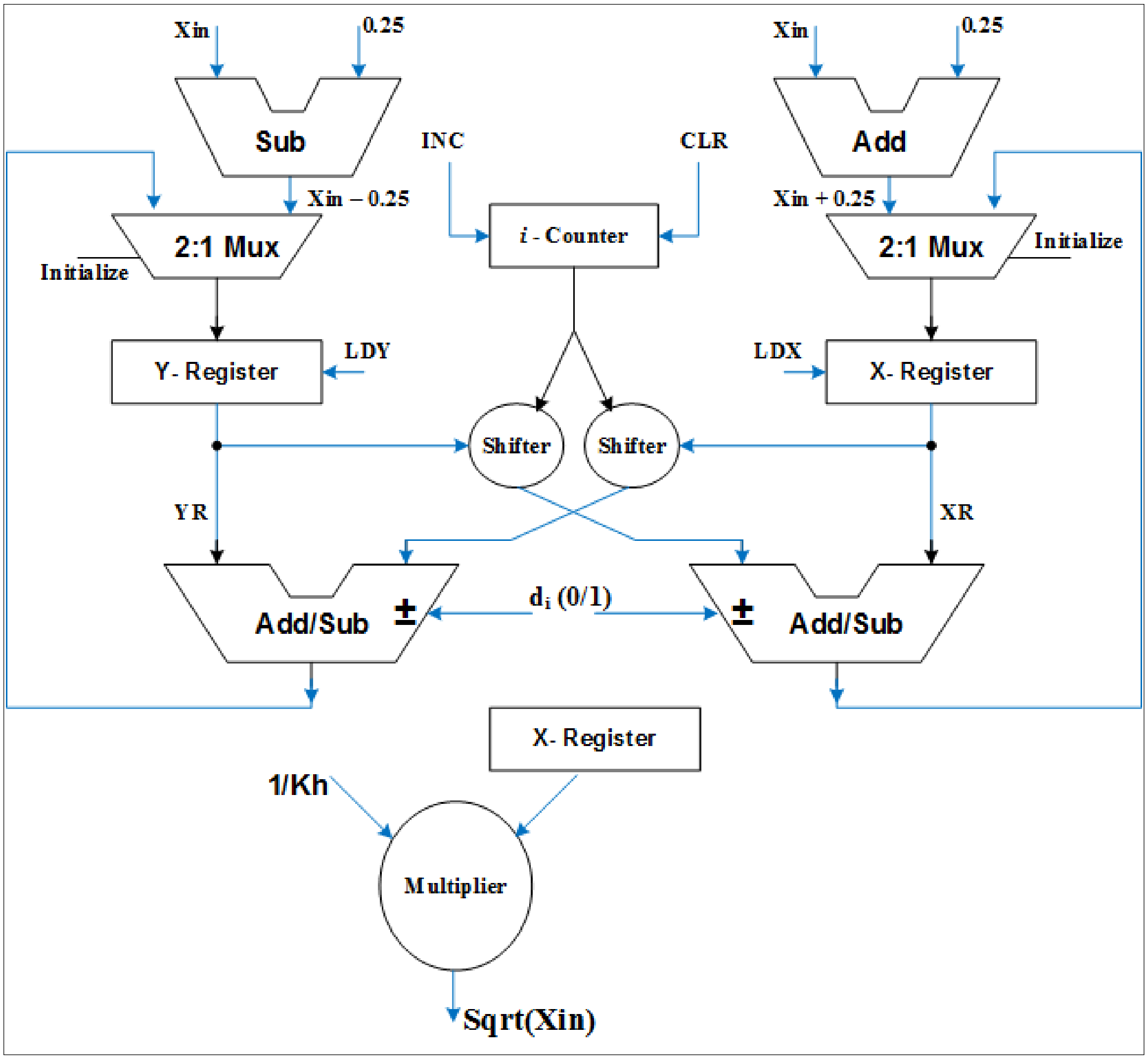


Figure 3.11: Data path of the square rooter using CORDIC

After n iterations (n is the size of input operands in bits) the register Y becomes zero while register X has value $K_h \times \sqrt{X_{in}}$ where K_h is the hyperbolic scale factor of CORDIC ($K_h = \prod_{i=1}^{n}(1/\cosh(2^{-i}))$) . Control signals like initialize, CLR (short for clear), INC (short for increment), LDX, and LDY (short for load x and y) are produced using a simple fsm controller.

Right after CORDIC module finishes its computations, the result in the X-register ($K_h \times \sqrt{X_{in}}$) is passed to a multiplier module which multiplies it with $1/K_h$ to produce $\sqrt{X_{in}}$ as shown in the data path.

Note that for square root calculations there is no need to update remaining angle (Z- path) of the CORDIC so we have excluded the remaining angle (Z-path) calculations from CORDIC processor to save the area. Further, to achieve an n-bit accuracy from any CORDIC processor we need to perform $n + log_2(n) + 2$ iterations of CORDIC with the same size input operands to get rid of round-off errors [66], [67] caused by shifts.

Tangent inverse function evaluator:

Tangent inverse module has been implemented using CORDIC processor to perform conversion from quaternions to Euler angles (equations(1.19) -(1.22)). CORDIC is used in vectoring mode with circular coordinates to compute tangent inverse function. Data path of the CORDIC-based atan2(y, x) function evaluating module is shown in Figure 3.12. It has 3 registers (X, Y, and Z), 1 counter, 4 add/subtract modules, 2 shifters and 4 multiplexers along with a read only memory (ROM) module to implement the four-quadrant atan2 function (equations(3.1)-(3.2)).

The counter is a mod n (n is the size of input operands in bits) counter which is initialized to 0 and is incremented by 1 (i=i+1) at rising edge of clock in every clock cycle. Using CORDIC, a total of n iterations/rotations of CORDIC rotator are performed to compute n bit result. Registers X and Y are initialized with the input operands X and Y respectively.

Shifters provide i-bit arithmetic shift right to X and Y register values. These shifted values of X and Y are, then, added/subtracted to/from Y-register and X-register respectively to update the value of X and Y registers in every clock cycle (equation(2.3)). The addition or subtraction operation depends on the value of d_i (d_i = Not(Sign(Y))).

The ROM stores pre-calculated values of micro rotation angle ($tan^{-1}(2^{-i})$). Its size, normally, is n×n bits (n is the size of input operands and same number of iterations/rotations of the CORDIC are performed to compute atan2 function). As suggested in [68], we have stored only $\frac{1}{3}n \times n$ bits in the ROM reducing its size to one-third of its original size. The micro rotation angle is selected from this ROM for only first one-third iterations while in the remaining two-third iterations, its approximated value 2^{-i} is used to update the Z-register.

The Z-register is initialized to 0 and is updated with micro rotation angle added/subtracted to the previous value of the Z-register every clock period. After n iterations it contains the value atan(y/x) which is then added to or subtracted from π to compute the four quadrant tangent inverse function atan2 (equation(3.1)) as shown in Figure 3.12.

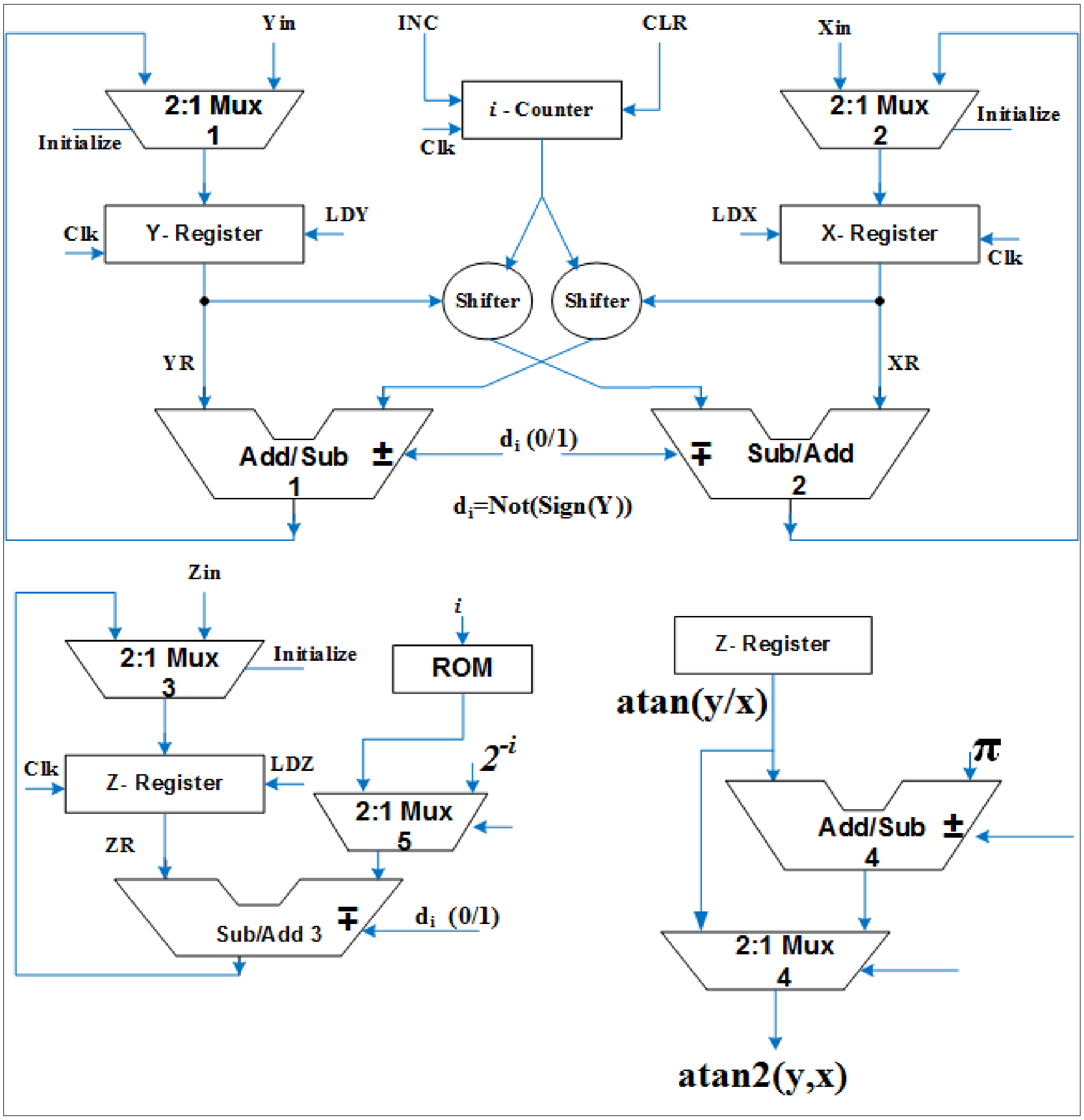


Figure 3.12: Data path of the CORDIC-based tangent inverse function evaluator.

CHAPTER 4

RESULTS AND DISCUSSIONS

The hardware of the all-accelerometer based IINS has been designed and modeled in VHDL using a combination of pipeline and iterative approaches to make it suitable for the guidance of high speed aerial vehicles (e.g. space shuttles and missiles) while having the minimum possible area. The model has been extensively simulated and tested for different input-output precisions.

Furthermore, it has been synthesized using CADENCE Encounter® RTL compiler with 90 nm Digital Standard Cell library. The VHDL generated results have been compared to the results produced by MATLAB. This chapter presents VHDL simulation and synthesis results.

4.1 Verification of VHDL generated results

Figure 4.1 shows angular velocities generated using our VHDL model (dotted lines), and the MATLAB generated data (dashed lines). The figure clearly shows that the VHDL generated angular velocities are almost in a perfect agreement with those generated by MATLAB.

Likewise, Figure 4.2 and Figure 4.3 compare the quaternion and attitude quantities generated by the VHDL model and those computed by MATLAB. Again, results from both figures demonstrate the correctness of the VHDL generated results and show that the results generated by VHDL model are faultlessly matching the results computed by MATLAB.

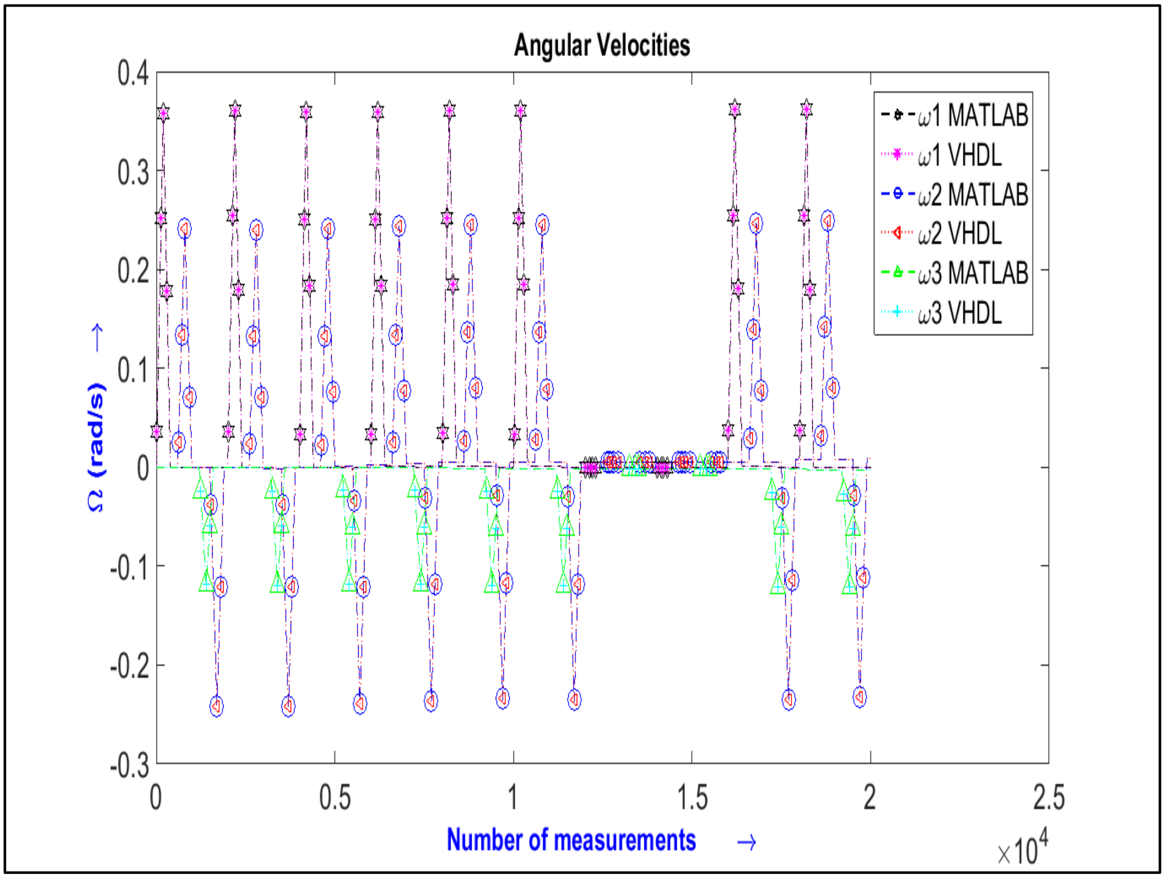


Figure 4.1: Comparison of angular velocities generated by VHDL and MATLAB.

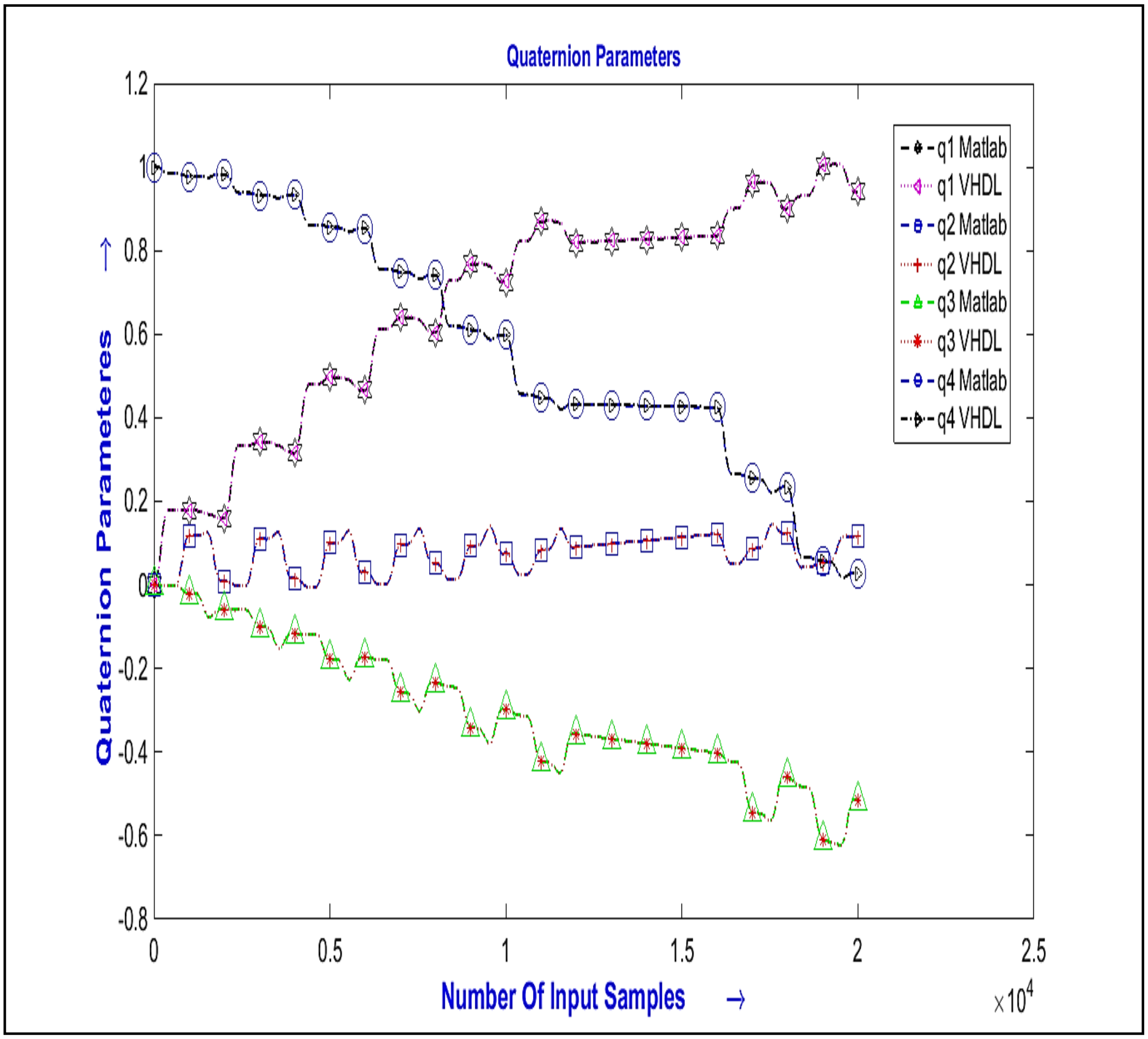


Figure 4.2: Comparison of quaternion parameters generated by VHDL and MATLAB

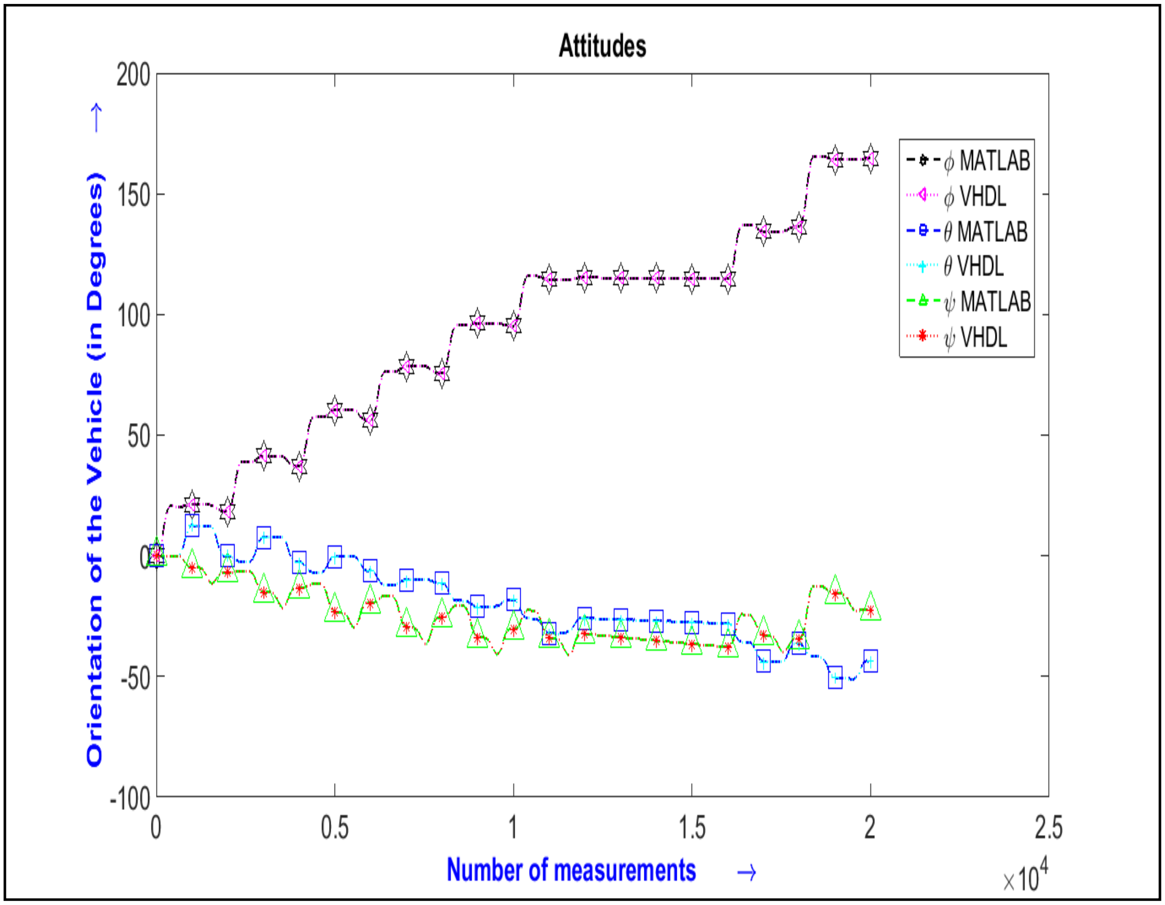


Figure 4.3: Orientation of the vehicle generated by the VHDL model and MATLAB.

4.2 Synthesis results

The proposed hardware of IINS has been synthesized using CADENCE Encounter® RTL compiler for a 90 nm Digital Standard Cell library. The resulting implementations' area, clock period, and latency are compared for different input-output precisions. These results are shown in the following table and graphs.

Table 4.1 : Synthesis results of the IINS processor

Number of Input/ Output bits	Clock period (ns)	# of Clocks required to do the job	Latency (ns)	Area (μ^2)	Area-latency product
16	1	54	54	492,962	26,619,948
24	2.9	78	226.2	645,066	145,913,929
32	4.99	102	508.98	827,304	421,081,190
40	7.24	126	912.24	976,799	891,075,120
48	9.3	150	1395	1,115,564	1,556,211,780
56	11.45	174	1992.3	1,265,059	2,520,377,046

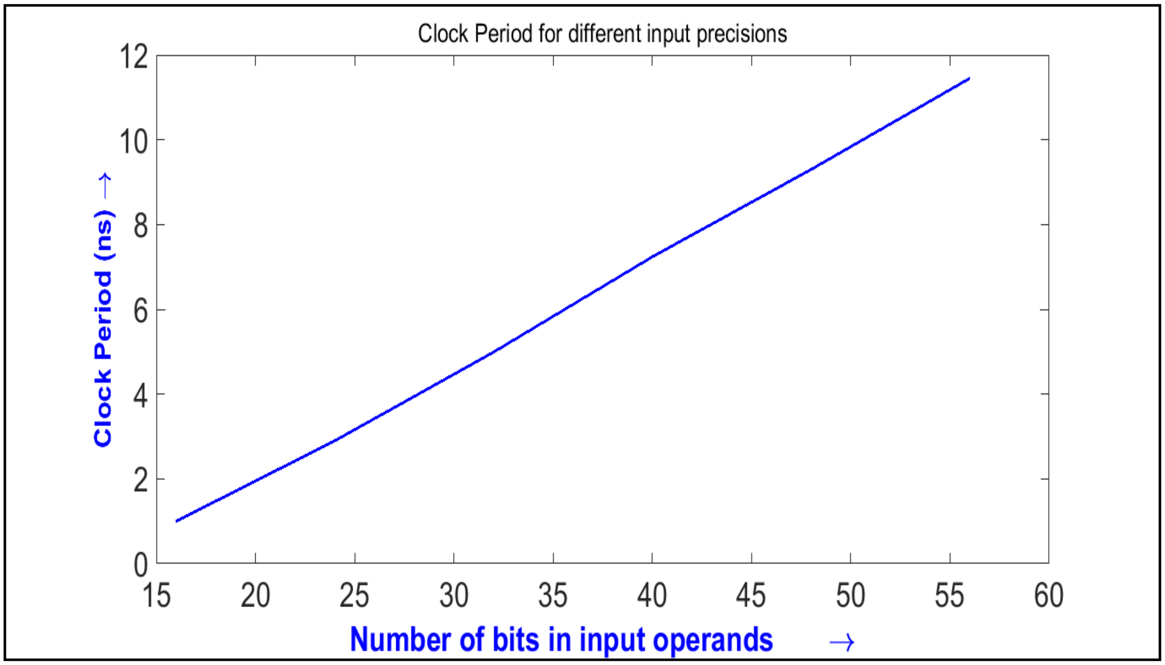


Figure 4.4: Clock period w.r.t no. of bits in input operands

Figure 4.4 portrays the trend of increase in clock period with increasing input precision. It is clear from the above figure, when the number of input bits of the design increases, clock period of the IINS model also increases linearly. It is due to the fact of using Carry Propagate (CP) adders in the hardware design which have delay of *O*(n) where n is the number of input bits. With larger input operands, adders' size increases and so does the delay of these larger adders. Furthermore, with larger input operands, the size of the overall design grows, and wire delays also become larger contributing to the longer clock periods. Figure 4.4 shows that the clock period gradually increases from 1 ns to 11.45 ns when input precision is increased from 16 bits to 56 bits.

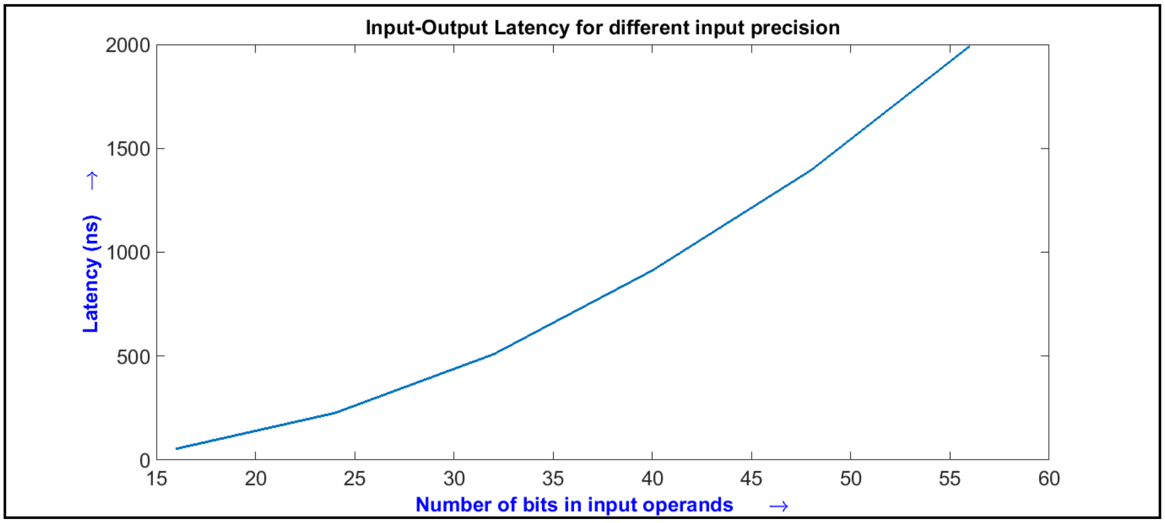


Figure 4.5: Input -to- output latency w.r.t no. of bits in input operands

Figure 4.5 to Figure 4.7 show the input-to-output latency, area, and area-latency product graphs respectively for the proposed IINS processor hardware. Latency is defined by the product of clock period and the number of clocks required to produce the output after input has been applied to the system, ignoring the time required to fill in the pipeline. For 16 bit input operands, for instance, the clock period is 1 ns, and an output is produced after every 54 clock cycles once the pipeline has been filled. It provides input-to-output latency of 54 ns for 16 bit input operands. The latency increases quadratically with the increase in input operands due to the usage of CP adders, and iterative implementations of the individual components (like, multipliers, dividers, square rooters, etc., which have a delay of $O(n^2)$ with CP adders) used in different stages of the pipeline. These iteratively implemented modules require more iterations for larger input operands to produce their output.

By the same token, larger size input operands demand larger internal registers, adders, and other components (like multipliers, dividers, square rooter, etc.) increasing the overall area and area-latency product of the design. This fact is depicted in Figure 4.6 and Figure 4.7.

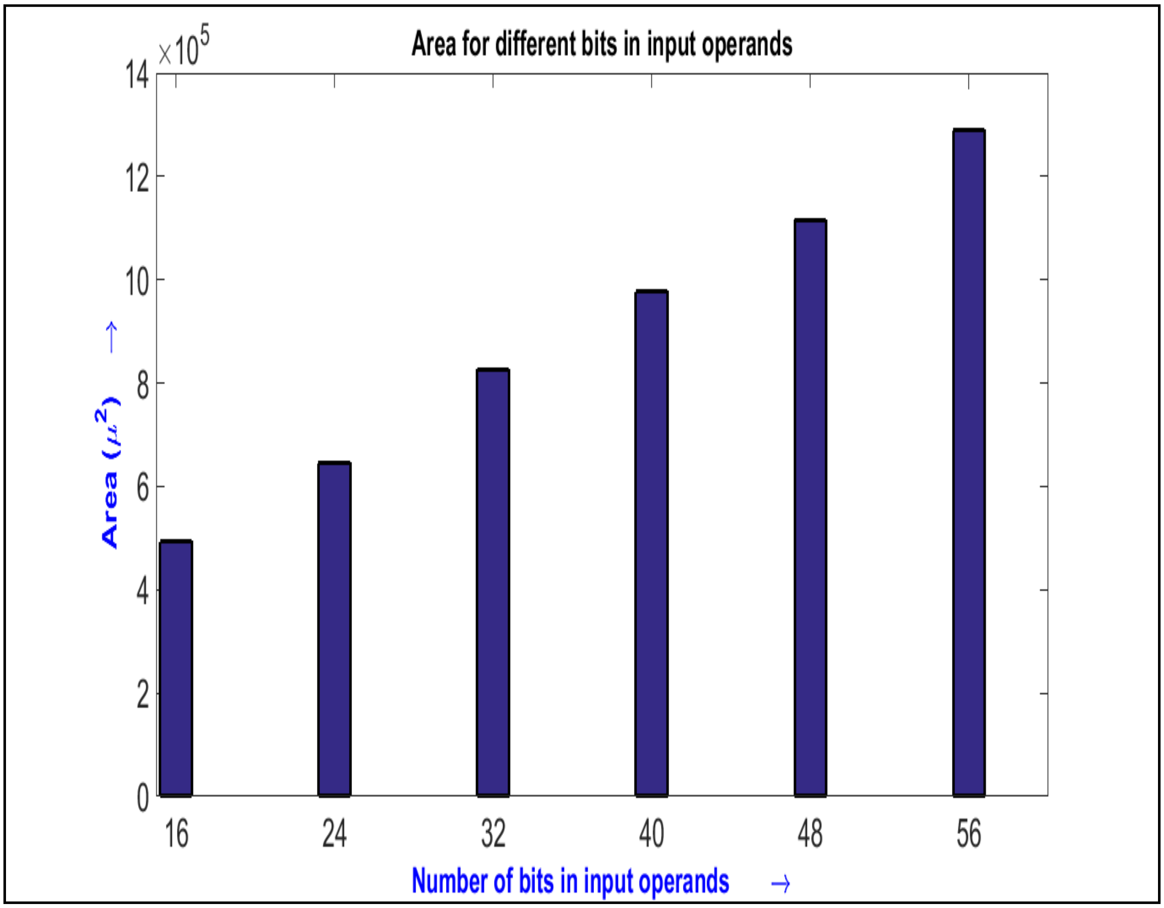


Figure 4.6: Area of the proposed processor for IINS w.r.t. no. of bits in input operands

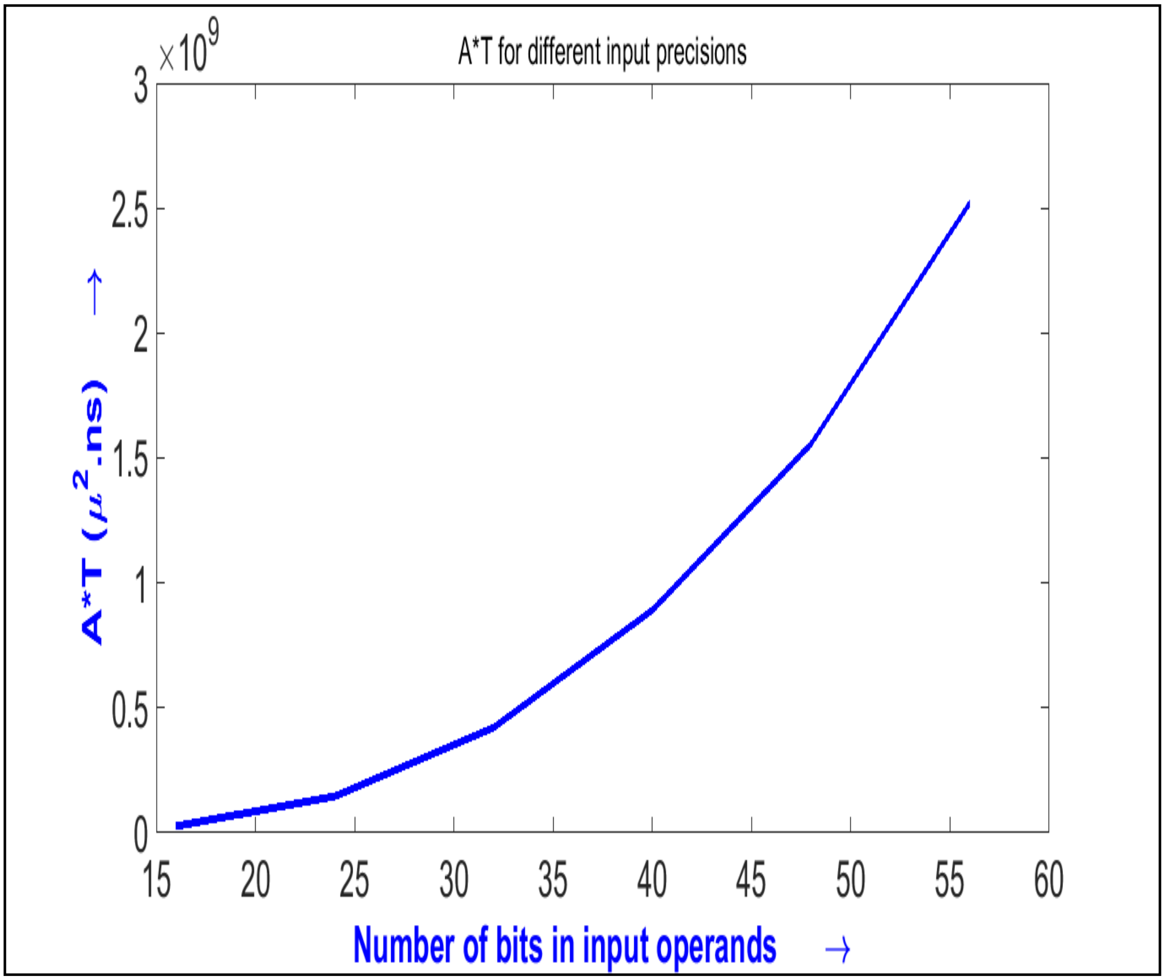


Figure 4.7: Area- latency product w.r.t. no. of bits in input operands.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Contributions

An application specific processor for an all-accelerometer based IINS system for aerial vehicles has been designed and modeled in VHDL. Architecture and design details of the proposed hardware of the IINS system have been provided. This hardware has been designed using a combination of pipeline and iterative approaches to make it suitable for the guidance of high speed aerial vehicles (e.g. Space shuttles and missiles) while having the minimum possible area. A working parameterized VHDL model of this system together with its test bench has been developed. A comparison of the VHDL generated results has been made to the results produced using MATLAB for verification purpose.

Furthermore, the proposed hardware model has been synthesized using CADENCE Encounter® RTL compiler for a 90 nm Digital Standard Cell library. The speed, latency, and area of the synthesized hardware have been evaluated for different input precisions. Produced results suggest that the designed IINS processor can operate at a frequency of 1 GHz with 54 ns input-to-output latency and an area of about 492,962 μ^2 for the 16 bit input operands. These results show that as the size of input operands increases, so does the area and input-to-output latency. Moreover, designs with larger input operands have longer clock periods because of longer wire delays and the *O*(n) delays caused by CP adders.

5.2 Future work

There are several promising research directions that can be pursued based on the results of this work. These improvements can be made in the following three domains:

Speed Critical applications:

For high speed applications Quaternion and DCM stages can be modified in such a way that instead of using 8 multipliers and multiplexers, these stages will use 16 multipliers to reduce their delay from the delay of two successive multipliers to that of a single multiplier.

1. A dedicated square rooter with less latency can be used instead of CORDIC-based square rooter in stage 8 of the pipeline.
2. Carry free adders (like Carry-Save, or BSD adders) can be used in the design to maintain a constant delay ($O(1)$) irrespective of the size of input operands. This will reduce the clock period and, in return, input-to-output latency.
3. More aggressively fast hardware can be designed by a fully pipelined and unrolled implementations of the individual components used in different pipeline stages.

The above mentioned modifications (any or all) can be implemented for speed demanding applications.

Area Critical Applications:

For area critical applications, the following modifications can be made to the IINS hardware.

1. Instead of using a mixture of pipelined and iterative design the hardware can be realized using a fully iterative approach using only 4 multipliers, 1 divider, 1 square rooter and 1 CORDIC atan function generator module.
2. For even slower applications, only a single multiplier, 1 divider, 1 square rooter, and 1 CORDIC processor along with a simple *fsm* controller are sufficient to implement the IINS processor. Moreover, the CORDIC module can be implemented in a way that only requires a single adder and few multiplexers to update the X, Y and Z variables.

Word length for different stages:

Fixed point computations with 32-bit word length have been used throughout the hardware implementation for the 16 bit input operands. Errors of different stages of the designed hardware have been computed. The angular velocity estimation stage, for instance, has a maximum error of 0.18%, whereas the quaternion stage has an error of 0.43%, and the maximum error of the orientation stage is 0.53%. All stages have a maximum error less than or equal to 0.5301%. Word length of different stages can be increased for more accuracy.

References

[1] "Gimball." [Online]. Available: http://www.globalspec.com/learnmore/sensors_transducers_detectors/tilt_sensing/inertial_gyros. [Accessed: 18-Apr-2015].

[2] R. P. G. Collinson, *Introduction to Avionics Systems*. London: Chapman and Hall, 1996.

[3] C.-W. Tan and S. Park, "Design of Accelerometer-Based Inertial Navigation Systems," *IEEE Trans. Instrum. Meas. [H.W. Wilson - AST]*, vol. 54, no. 6, 2005.

[4] M. F. Almalki and M. Elshafei, "Method and Apparatus for tracking center of gravity of air vehicle," *J. Eng.*, no. U. S. Patent, US8260477 B2, September 4, 2012.

[5] Y. M. Al-rawashdeh, "Improved Inertial Navigation System using All-Accelerometers ,MSc thesis.," King Fahad University of Petroleum and Minerals, 2014.

[6] G. Hyslop, "A Norm and Orthogonality Preserving Algorithm," *IEEE Trans. Aerosp. Electron. Syst.*, vol. AES-23, no. 6, pp. 731–737, 1987.

[7] J. E. Volder, "The CORDIC Trigonometric Computing Technique," *Electron. Comput. IRE Trans.*, vol. EC-8, no. 3, 1959.

[8] J. S. Walther, "A unified algorithm for elementary functions," *Proc. May 18-20, 1971, spring Jt. Comput. Conf. - AFIPS '71*, no. 9, p. 379, 1971.

[9] J. Mcleod, "Spec Puts Video Conferencing Closer to Desktop," *Electronics*, vol. 68.

[10] M. Noore. A, Nestor S, Lawson, "Computer-based multimedia video conferencing system," *IEEE Trans. Consum. Electron.*, vol. 39, no. 3, pp. pp.587–592.

[11] H. M. Ahmed, "Efficient elementary function generation with multipliers," *Proc. 9th Symp. Comput. Arith.*, 1989.

[12] H. M. Ahmed, "Signal Processing Algorithms and Architectures," Stanford University.

[13] D. Timmermann, H. Hahn, and B. Hosticka, "Erratum: Modified CORDIC algorithm with reduced iterations," *Electron. Lett.*, vol. 25, no. 17, p. 1201, 1989.

[14] M. D. Ercegovac and T. Lang, “Redundant and on-line CORDIC: Application to matrix triangularization and SVD,” *IEEE Trans. Comput.*, vol. 39, no. 6, pp. 725–740, 1990.

[15] E. Antelo, J. D. Bruguera, and E. L. Zapata, “Unified mixed radix 2-4 redundant CORDIC processor,” *IEEE Trans. Comput.*, vol. 45, no. 9, pp. 1068–1073, 1996.

[16] H. Dawid and H. Meyr, “The differential CORDIC algorithm: Constant scale factor redundant implementation without correcting iterations,” *IEEE Trans. Comput.*, vol. 45, no. 3, pp. 307–318, 1996.

[17] J. Duprat and J. M. Muller, “CORDIC algorithm. New results for fast VLSI implementation,” *IEEE Trans. Comput.*, vol. 42, no. 2, pp. 168–178, 1993.

[18] K. Maharatna, S. Banerjee, E. Grass, M. Krstic, and A. Troya, “Modified virtually scaling-free adaptive CORDIC rotator algorithm and architecture,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 11, pp. 1463–1473, 2005.

[19] K. Maharatna, A. Troya, S. Banerjee, and E. Grass, “Virtually scaling-free adaptive CORDIC rotator,” *IEE Proceedings - Computers and Digital Techniques*, vol. 151, no. 6. p. 448, 2004.

[20] Haviland and Tuszynski, “A Cordic Arithmetic Processor Chip,” *IEEE Transactions on Computers*, vol. C–29, no. 2. pp. 68–79, 1980.

[21] H. Hahn, D. Timmermann, B. J. Hosticka, and B. Rix, “Unified and division-free CORDIC argument reduction method with unlimited convergence domain including inverse hyperbolic functions,” *IEEE Trans. Comput.*, vol. 43, no. 11, pp. 1339–1344, 1994.

[22] S. S. Nikolaidis, D. E. Metafas, and C. E. Goutis, “CORDIC based pipeline architecture for all-pass filters7,” *1993 IEEE Int. Symp. Circuits Syst.*, 1993.

[23] G. J. Hekstra and E. F. A. Deprettere, “Floating point Cordic,” *Proc. IEEE 11th Symp. Comput. Arith.*, 1993.

[24] S. F. Hsiao and J. M. Delosme, “Householder CORDIC algorithms,” *IEEE Trans. Comput.*, vol. 44, no. 8, pp. 990–1001, 1995.

[25] S.-F. H. S.-F. Hsiao and J.-M. Delosme, “Parallel singular value decomposition of complex matrices using multidimensional CORDIC algorithms,” *IEEE Trans. Signal Process.*, vol. 44, no. 3, 1996.

[26] B. Rashidi, B. Rashidi, and M. Pourormazd, “Design and implementation of low power digital FIR filter based on low power multipliers and adders on xilinx FPGA,” *2011 3rd Int. Conf. Electron. Comput. Technol.*, vol. 2, pp. 18–22, 2011.

[27] S. Shah, A. J. Al-Khalili, and D. Al-Khalili, “Comparison of 32-bit multipliers for various performance measures,” *ICM 2000. Proc. 12th Int. Conf. Microelectron. (IEEE Cat. No.00EX453)*, 2000.

[28] S. Shah, “Design and Comparison of 32-bit Multipliers for Various Performance Measures,” Concordia University.

[29] O. Salomon, J.-M. Green, and H. Klar, “General Algorithm for a Simplified Addition of 2’s Complement Numbers in Multipliers,” *Solid-State Circuits Conf. 1994. ESSCIRC ’94. Twent. Eur.*, 1994.

[30] A. D. Booth, “A signed binary multiplication technique,” *Q. J. Mech. Appl. Math.*, vol. 4, no. 2, pp. 236–240, 1951.

[31] C. S. Wallace, “A Suggestion for a Fast Multiplier,” *IEEE Trans. Electron. Comput.*, vol. EC-13, no. 1, 1964.

[32] P. Larsson-Edefors, “A 965-Mb/s 1.0-??m standard CMOS twin-pipe serial/parallel multiplier,” *IEEE J. Solid-State Circuits*, vol. 31, no. 2, pp. 230–239, 1996.

[33] L. Dadda, “Some Schemes for Parallel Multipliers,” *Alta Freq.*, vol. 34, pp. 349 – 356.

[34] W. J. Townsend, E. E. Swartzlander Jr, and J. A. Abraham, “A comparison of Dadda and Wallace multiplier delays,” *Adv. Signal Process. Algorithms, Archit. Implementations XIII. Proc. SPIE*, 2003.

[35] C. Y. H. Lee, L. H. Hiung, S. W. F. Lee, and N. H. Hamid, “A performance comparison study on multiplier designs,” in *2010 International Conference on Intelligent and Advanced Systems, ICIAS 2010*, 2010.

[36] D. Ferrari, “A Division Method Using a Parallel Multiplier,” *IEEE Trans. Electron. Comput.*, vol. EC-16, no. 2, 1967.

[37] M. J. Flynn, “On Division by Functional Iteration,” *IEEE Trans. Comput.*, vol. C–19, no. 8, 1970.

[38] R. E. Goldsmith, “Applications of division by convergence,” MIT.

[39] J. E. Robertson, “A New Class of Digital Division Methods,” *Electron. Comput. IRE Trans.*, vol. EC-7, no. 3, 1958.

[40] N. Aggarwal, K. Asooja, S. S. Verma, and S. Negi, “An improvement in the restoring division algorithm: Needy restoring division algorithm,” in *Proceedings -*

2009 2nd IEEE International Conference on Computer Science and Information Technology, ICCSIT 2009, 2009, pp. 246–249.

[41] M. D. Ercegovac and T. Lang, “A division algorithm with prediction of quotient digits,” *1985 IEEE 7th Symp. Comput. Arith.*, pp. 51–56, 1985.

[42] K. S. Trivedi and M. D. Ercegovac, “On-line algorithms for division and multiplication,” *1975 IEEE 3rd Symp. Comput. Arith.*, pp. 161–167, 1975.

[43] M. D. Ercegovac and T. Lang, “On-line arithmetic for DSP applications,” *Proc. 32nd Midwest Symp. Circuits Syst.*, no. August 1989, pp. 0–3, 1989.

[44] F. P. Preparata and J. E. Vuillemin, “Practical cellular dividers,” *IEEE Trans. Comput.*, vol. 39, no. 5, pp. 605–614, 1990.

[45] M. D. Ercegovac and T. Lang, *Digital Arithmetic*. MORGAN KAUFMANN, 2003.

[46] Y. L. Y. Li and W. C. W. Chu, “Implementation of single precision floating point square root on FPGAs,” *Proceedings. 5th Annu. IEEE Symp. Field-Programmable Cust. Comput. Mach. Cat. No.97TB100186)*, 1997.

[47] J. Detrey and F. De Dinechin, “A tool for unbiased comparison between logarithmic and floating-point arithmetic,” *J. VLSI Signal Process. Syst. Signal Image. Video Technol.*, vol. 49, no. 1, pp. 161–175, 2007.

[48] D. M. Russinoff, “Mechanically checked proof of correctness of the AMD K5 floating point square root microcode,” *Form. Methods Syst. Des.*, vol. 14, no. 1, pp. 75–125, 1999.

[49] C. P. Jeannerod, H. Knochenl, C. Monat, and G. Revy, “Faster floating-point square root for integer processors,” in *2007 Symposium on Industrial Embedded Systems Proceeedings, SIES’2007*, 2007, pp. 324–327.

[50] J. A. Piñeiro and J. D. Bruguera, “High-speed double-precision computation of reciprocal, division, square root, and inverse square root,” *IEEE Trans. Comput.*, vol. 51, no. 12, pp. 1377–1388, 2002.

[51] P. Behrooz, *Computer arithmetic: Algorithms and hardware designs*, 1st ed. New York, USA: Oxford University Press, 2000.

[52] “atan2 - C++ Reference.” [Online]. Available: http://www.cplusplus.com/reference/cmath/atan2/. [Accessed: 04-May-2015].

[53] “Four-quadrant inverse tangent - MATLAB atan2,” 2014. [Online]. Available: http://www.mathworks.com/help/matlab/ref/atan2.html?searchHighlight=atan2. [Accessed: 04-May-2015].

[54] “ArcTan—Wolfram Language Documentation.” [Online]. Available: http://reference.wolfram.com/language/ref/ArcTan.html. [Accessed: 04-May-2015].

[55] “Math (Java Platform SE 6).” [Online]. Available: http://docs.oracle.com/javase/6/docs/api/java/lang/Math.html. [Accessed: 04-May-2015].

[56] D. D. Hwang, D. F. D. Fu, and A. N. . J. Willson, “A 400-MHz processor for the efficient conversion of rectangular to polar coordinates for digital communications applications,” *2002 Symp. VLSI Circuits. Dig. Tech. Pap. (Cat. No.02CH37302),* 2002.

[57] W. F. Wong and E. Goto, “Fast hardware-based algorithms for elementary function computations using rectangular multipliers,” *IEEE Trans. Comput.*, vol. 43, no. 3, pp. 278–294, 1994.

[58] M. R. D. Rodrigues, J. H. P. Zurawski, and J. B. Gosling, “Hardware evaluation of mathematical functions,” *IEE Proc. E Comput. Digit. Tech.*, vol. 128, no. 4, p. 155, 1981.

[59] I. Koren, *Computer Arithmetic Algorithms*, 2nd ed. A.K.Peters, 2002.

[60] R. G. Lyons, “Another Contender in the Arctangent Race,” *Streamlining Digit. Signal Process. A Tricks Trade Guideb. Second Ed.*, no. January, pp. 237–242, 2012.

[61] I. Koren and O. Zinaty, “Evaluating elementary functions in a numerical coprocessor based on rational approximations,” *IEEE Trans. Comput.*, vol. 39, no. 8, pp. 1030–1037, 1990.

[62] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fourth Edition: A Quantitative Approach*, no. 0. 2006.

[63] S. S. Leung and M. A. Shanblatt, *ASIC system design with VHDL, a paradigm*. Kluwer Academic Publishers, 1989.

[64] E. I. Organick, *A FORTRAN IV Primer*. Addison-Wesley, 1966.

[65] Ercegovac and Lang, “On-the-Fly Conversion of Redundant into Conventional Representations,” *IEEE Transactions on Computers*, vol. C–36, no. 7. pp. 895–897, 1987.

[66] E. Antelo, J. D. Bruguera, T. Lang, and E. L. Zapata, “Error analysis and reduction for angle calculation using the CORDIC algorithm,” *IEEE Trans. Comput.*, vol. 46, no. 11, pp. 1264–1271, 1997.

[67] Y. H. Hu, “The quantization effects of the CORDIC algorithm,” *IEEE Trans. Signal Process.*, vol. 40, no. 4, pp. 834–844, 1992.

[68] N. Takagi, T. Asada, and S. Yajima, “Redundant CORDIC methods with a constant scale factor for sine and cosine computation,” *IEEE Trans. Comput.*, vol. 40, no. 9, 1991.

Vitae

Name: Muhammad Ijaz

Nationality: Pakistani

Date of Birth: 15/10/1985

Email: mijaz@ciitlahore.edu.pk

Address: Hasilpur, Bahawalpur, Pakistan

Academic Background: MS in Computer Engineering, 2015, King Fahd University of Petroleum & Minerals, Saudi Arabia.

BS in Electrical (Telecom) Engineering, 2009, CIIT, Lahore Pakistan.

Publications: High-Radix Modulo-Multiplication Using a Scaled Modulus, IEEE Transactions on Very Large Scale Integrated Systems, (Submitted).

An ASIC processor for all-accelerometer based Integrated Inertial Navigation System for Aerial Vehicles (In progress).