12-15-2016

Implementing Partial Blind Estimation of Alamouti Space-Time Coding in Software-Defined Radios

Sai Gautham Peesapati
sp5584@rit.edu

R·I·T

Implementing Partial Blind Estimation of Alamouti Space-Time Coding in Software-Defined Radios

by

Sai Gautham Peesapati

December 15, 2016

Faculty Advisor: Prof. Miguel Bazdresch

Thesis committee

Drew Maywar, Associate Professor

Steven Ciccarelli, Associate Professor

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Telecommunications Engineering Technology.

Electrical, Computer and Telecommunications Engineering Technology

College of Applied Science and Technology

Rochester Institute of Technology

Rochester, NY

Committee Approval

Dr. Miguel Bazdresch Date

Thesis Advisor

Dr. Drew Maywar Date

Committee Member

Steven Ciccarelli Date

Committee Member

Abstract

The aim of this thesis is to develop and study the performance of an algorithm to perform blind estimation of Alamouti space-time coding (STC) on Zedboard's ARM core and FPGA, and propose improvements based on the collected data.

In the first part, previous methods to accomplish blind estimation of Alamouti STC are presented, and a new method that is simple enough for hardware implementation is proposed. The system of linear equations in the final step of the proposed method is not mathematically solvable. Hence, in the next part, the thesis proposes a method to accomplish partial blind estimation of Alamouti STC, that is suitable for fixed-point implementation. The new method for partial blind estimation is then implemented on ARM core and FPGA. The FPGA implementation is performed in three modes - power optimization, runtime optimization and area optimization.

In conclusion, the thesis draws common observations from the data collected from simulations, ARM and FPGA implementations of the proposed method, to find its drawbacks. The performance data from the two hardware implementations are studied to compare the two methods, and find their advantages and disadvantages. This thesis hopes to offer researchers in the area of blind estimation, what is needed to develop a feasible method of accomplishing blind estimation of Alamouti STC, and what is to be expected from implementing the same on hardware.

Contents

Chapter 1

Introduction

This thesis studies the process of selection of a blind estimation algorithm for a system implementing Alamouti space-time coding (STC) and implementation on ARM and FPGA devices. Blind estimation algorithms for Alamouti STC exist, but have not yet been implemented in an existing wireless communication system [3, 5, 6, 9, 10, 11, 13].

The background about Alamouti STC and blind estimation of the scheme is given in chapter 1. The modification and explanation of the implemented blind estimation algorithm is given in chapter 2. The hardware architecture to be implemented in FPGA is given in chapter 3. The performance of the blind estimation algorithm implemented in C code on ARM is studied in chapter 4. The performance of the blind estimation algorithm implemented on FPGA using Xilinx System Generator is studied in chapter 5. The MATLAB, C and VHDL codes used for hardware implementation are given in the appendices.

1.1 Definition of Symbols Used

1. t=Time.
2. $u(t)$= Baseband transmitted signal in the time-domain.
3. $s(t)$= Upconverted baseband transmitted signal in the time-domain.
4. f_c= Carrier Frequency.
5. ω_0=Phase of the carrier.
6. $r(t)$= Upconverted baseband received signal in the time-domain.
7. $v(t)$= Baseband received signal in the time-domain.
8. N=Number of independent paths taken by the signal to reach the receiver.
9. ψ_n=Power gain along the n^{th} path of propagation.
10. ω_n=Phase shift introduced due to propagation along the n^{th} path.
11. τ_n=Time taken by the n^{th} path to travel from the transmitter to the receiver.
12. T=Symbol duration.
13. s_1=First transmitted symbol in the Alamouti scheme.
14. s_2=Second transmitted symbol in the Alamouti scheme.
15. h_1=First channel coefficient in the Alamouti scheme.
16. h_2=Second channel coefficient in the Alamouti scheme.
17. x_1=First received symbol in the Alamouti scheme.
18. x_2=Second received symbol in the Alamouti scheme.

19. n_1=Additive white Gaussian noise in the first received symbol.

20. n_2=Additive white Gaussian noise in the second received symbol.

21. $E[]$=Expected value operator.

22. X=Received symbol matrix = $\begin{bmatrix} x_1 \\ x_2^* \end{bmatrix}$.

23. H=Channel coefficient matrix = $\begin{bmatrix} h_1 & h_2 \\ h_2^* & -h_1^* \end{bmatrix}$.

24. R_s=Correlation matrix of the transmitted symbols.

25. σ_n=Variance of the noise in the communication system.

26. I=Identity matrix of order 2 = $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

27. $||h||$=Sum of magnitudes of channel coefficients.

28. γ=Ratio of powers of the two symbols in the Alamouti scheme, where the power of the symbol is the square of the voltage induced by the transmitted signal.

29. σ_s=Power associated with s_1.

30. $\sigma_h^2 = \frac{\sigma_n^2}{\sigma_s^2 ||h||^2}$.

31. s_{1i}=First symbol transmitted during the i^th Alamouti time-frame.

32. s_{2i}=Second symbol transmitted during the i^th Alamouti time-frame.

33. $s_{1(i+1)}$=First symbol transmitted during the $(i+1)^th$ Alamouti time-frame.

34. $s_{2(i+1)}$=Second symbol transmitted during the $(i+1)^th$ Alamouti time-frame.

35. z_{1i}=First symbol received during the i^th Alamouti time-frame.

36. z_{2i}=Second symbol received during the i^th Alamouti time-frame.

37. $z_{1(i+1)}$=First symbol received during the $(i+1)^th$ Alamouti time-frame.

38. $z_{2(i+1)}$=Second symbol received during the $(i+1)^th$ Alamouti time-frame.

39. r=Ratio of average powers of the 2 transmit constellations.

40. $V = \begin{bmatrix} 1 & 0 \\ 0 & r \end{bmatrix}$.

41. ϕ_1=Phase shift introduced by the first channel in the Alamouti scheme.

42. ϕ_2=Phase shift introduced by the second channel in the Alamouti scheme.

43. α_1=Phase of the first symbol transmitted in Alamouti STC

44. α_2=Phase of the second symbol transmitted in Alamouti STC

1.2 Background

The propagation of electromagnetic waves in a medium can be studied using ray-tracing techniques, which assume that these waves travel in the form of multiple rays being reflected, scattered and diffracted by objects in the vicinity of the transmitter and the receiver. Each ray has a attenuation, phase shift and a time delay that it introduces to the transmitted signal. When a signal is transmitted, it gets radiated in all directions at the same time. Hence, the receiver receives multiple instances of the same signal. These instances can be either spread out over time or aggregated to form a single resultant signal. Note that the signals are considered to be complex numbers to simplify calculations in the complex frequency-domain. The amplitude of the complex number is the voltage induced by the electromagnetic signal and the

phase is the phase delay of the signal. The transmitted signal is modeled as [2]

$$s(t) = R\{u(t)e^{j(2\pi f_c t+\omega_0)}\}.$$

The received signal is modeled as [2]

$$r(t) = R\{v(t)e^{j(2\pi f_c t+\omega_0)}\}.$$

As the received signal in a propagation environment with rich scattering, is the net result of all the independent rays reaching the receiver the baseband received signal can be written as

$$v(t) = \sum_{1}^{N}(\sqrt{\psi_n}e^{j\omega_n}u(t-\tau_n)).$$

where N is the number of independent paths taken by the transmitted signal, ψ_n is the power attenuation along the n^{th} path, ω_n is the phase shift along the n^{th} path, and τ_n is the propagation delay along the n^{th} path.

When the transmitted signal $u(t)$ is an impulse, the received signal is known as the channel impulse response of the system. The channel impulse response at time t is given by

$$c(t) = \sum_{1}^{N}(\sqrt{\psi_n(t)}e^{j\omega_n(t)}\delta(t-\tau_n(t))).$$

where $\psi_n(t)$ is the power attenuation along the n^{th} path at time t, $\omega_n(t)$ is the phase shift along the n^{th} path at time t, and $\tau_n(t)$ is the propagation delay along the n^{th} path at time t.

In the continuous time-domain, the received signal is given by

$$v(t) = c(t) * u(t) = \int_{-\infty}^{\infty} c(\tau)u(t-\tau)d\tau.$$

Every wireless communication system is characterized by three parameters, which are:

- **Signal Bandwidth (B)** is the reciprocal of the transmitted symbol.

- **Delay Spread (τ_m)** is the difference between the propagation delays of the first and last component of the received signal.

- **Coherence Time (τ_c)** is the time duration over which the impulse response of the channel remains constant.

Two signal components from two independent paths are said to be completely independent and resolvable if $\triangle\tau >> B^{-1}$, where B is the signal bandwidth and $\triangle\tau$ is the difference between the propagation times of the two components. For signals received by two antennas to be independent, the separation between the two antennas must be determined by pretesting the system in its specific propagation environment. A good approximation of the minimum separation between the antennas to obtain independent signals is 0.37 times the wavelength of the carrier frequency.

When $\tau_m >> B^{-1}$, the channel is known as a wide-band channel, and the individual paths can be resolved. In this type of channel, the spectrum of the received signal is considerably different from that of the transmitted signal. Hence, propagation in this kind of channel is also known as frequency-selective fading as the channel acts like a filter. When $\tau_m << B^{-1}$, the channel is known as a narrow-band channel, and the individual paths cannot be resolved. In this type of channel, the spectrum of the received signal is negligibly different from that of the transmitted signal. Hence,

propagation in this kind of channel is also known as flat fading (the channel's frequency response is flat) and the channel acts like an attenuator [2].

When $\tau_m << \tau_c$, the amplitude and the phase change due to the channel vary considerably multiple times over the period of transmission of a signal. This situation is known as fast fading. When $\tau_m >> \tau_c$, the channel stays roughly constant over the period of transmission of a signal. This situation is known as slow fading [2]. In a slow fading situation $c(\tau)$ is constant. The bit-error rate in a communication system [2] is given by

$$P_b = 0.5\left(erfc\left(\sqrt{\frac{E_b}{N_0}}\right)\right)$$

where E_b is the energy per bit and N_0 is the noise energy in the system.

This thesis shall assume a slow-flat fading system, from hereon. In slow-flat fading, the receiver receives only one instance of the transmitted signal, which is attenuated and phase-shifted, but with the same pulse shape. Hence, the baseband received signal is given by

$$v(t) = \sqrt{\psi}e^{j\omega}u(t-\tau) = hu(t-\tau).$$

The model of a slow-flat fading system in the discrete time-domain is

$$r = hs$$

where r and s are the received and transmitted signals respectively, and h is a complex random variable the amplitude of which has a Rayleigh distribution.

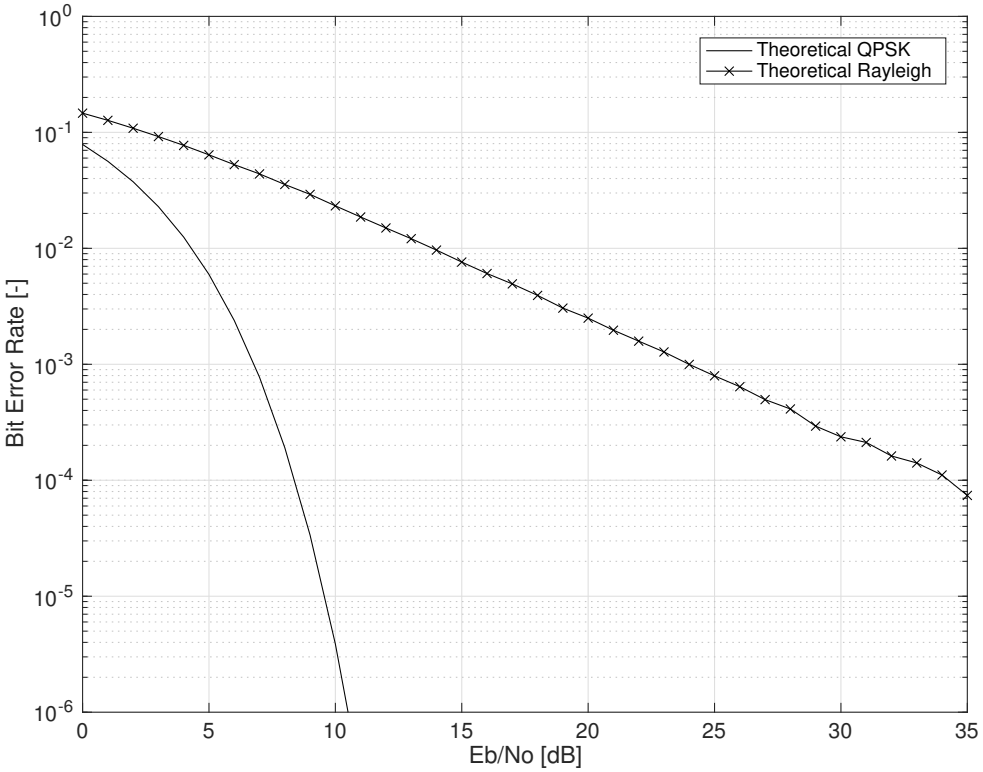

Figure 1.1: Comparison of bit-error rates of a QPSK constellation in Rayleigh and AWGN channels.

1.3 Diversity

Diversity [2, 4] is a scheme used in wireless communication systems in which multiple copies of a signal are sent or received to improve the performance of the system. Each copy of a signal experiences independent or close-to-independent fading. When multiple copies of the same signal are aggregated at the receiver, the probability of receiving an unfavorable copy decreases with an increase in the number of signal instances. The techniques to combine the multiple received signals [4], to achieve a better bit error rate are:

- In **Selection combining** [2, 4], the receiver selects the signal copy with the highest SNR.

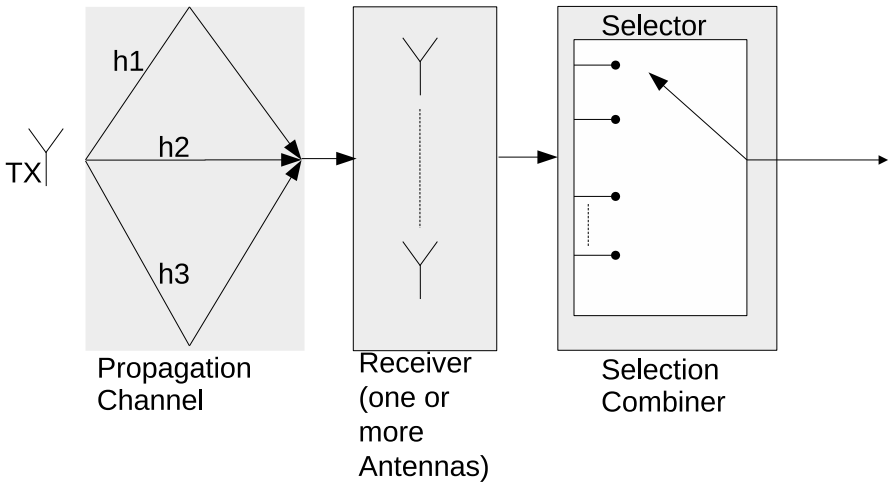

Figure 1.2: Block diagram of selection combining

- In **linear combining** [2, 4], the receiver nullifies the phase change due to the channel and adds all the copies together. The output signal is given by

$$r = (\sum h_n s e^{-\phi_n}),$$

$$r = (\sum |h_n| e^{\phi_n} e^{-\phi_n}) s,$$

$$r = (\sum |h_n|) s$$

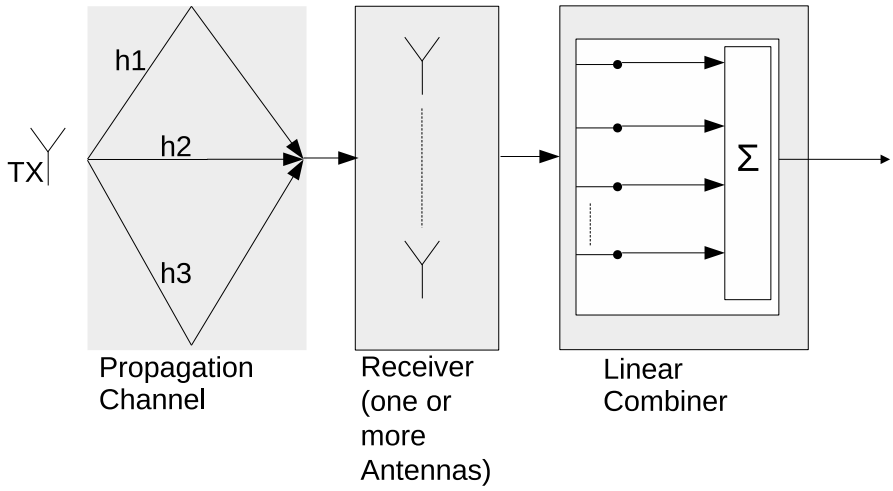

Figure 1.3: Block diagram of linear combining. Nullification of channel coefficient phase not shown.

- In **maximum ratio combining** (MRC) [2, 4], each signal copy is multiplied by a different gain so that a maximum net SNR is obtained. The copies are multiplied with gains directly proportional to their respective SNRs to give more weight to the better ones. The resulting SNR of the combiner increases approximately linearly with the number of diversity branches. The output signal is given by

$$r = \sum (w_i h_i s + w_i n_i),$$

where n_i is the instantaneous noise in the i^{th} branch.

If we assume that transmitted signal has unit average power, The output SNR of the combiner is

$$SNR_{out} = \frac{(\sum w_i h_i)^2}{E[(w_i n_i)^2]}.$$

If the average power of the noise in the system is σ^2 and $\frac{h_i^2}{\sigma^2}$ is the SNR in the i^{th} channel,

$$SNR_{out} = N\frac{E[w_i h_i]^2}{\sigma^2 E[w_i^2]}.$$

The above equation has a maximum value when $w_i = h_i$,

$$SNR_{out} = N\frac{E[h_i^2]^2}{\sigma^2 E[h_i^2]},$$

$$SNR_{out} = N\frac{E[h_i^2]}{\sigma^2},$$

$$SNR_{out} = N.E[SNR_i]$$

and

$$SNR_{out} = \sum SNR_i.$$

Maximum ratio combining has the best performance of all the diversity combination schemes.

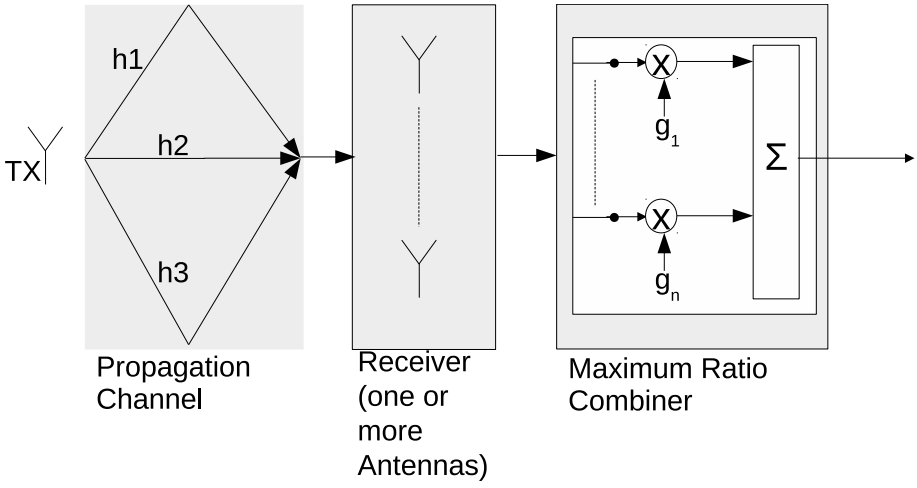

Figure 1.4: Block diagram of maximum ratio combining

- In **equal gain combining** [2, 4], all the signal copies are multiplied by the

same gain and aggregated. The output signal is given by

$$r = \sum (wh_i s + wn_i).$$

If we assume that transmitted signal has unit average power, The output SNR of the combiner is

$$SNR_{out} = \frac{(\sum wh_i)^2}{E[(wn_i)^2]}.$$

If the average power of the noise in the system is σ^2 and $\frac{h_i^2}{\sigma^2}$ is the SNR in the i^{th} channel,

$$SNR_{out} = N\frac{E[wh_i]^2}{\sigma^2 E[w^2]},$$

$$SNR_{out} = Nw^2\frac{E[h_i]^2}{\sigma^2 w^2}$$

and

$$SNR_{out} = N.E[SNR_i].$$

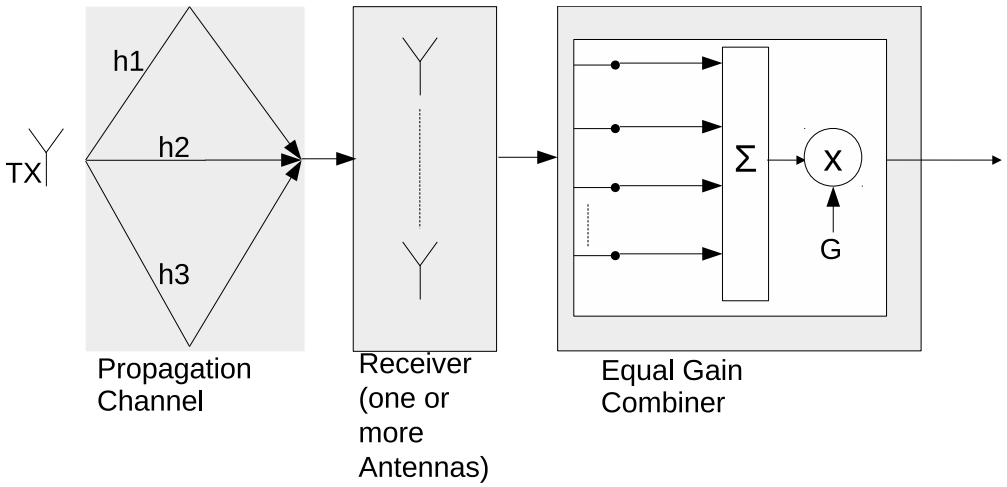

Figure 1.5: Block diagram of equal gain combining

- In **hybrid combining** [2, 4], two or more of the above methods are combined to further optimize the SNR.

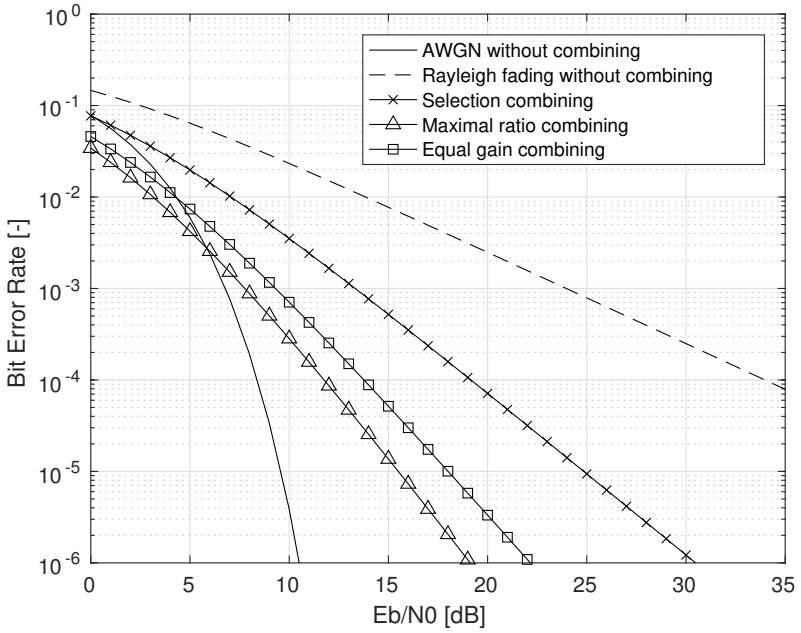

Figure 1.6: Comparison of theoretical bit-error rates of the diversity combination techniques with 3 diversity branches, in a QPSK system. Linear combining is not shown because it is equivalent to equal-gain combining with unit gain. The above graph shows that maximal-ratio combining (MRC) has the highest decrease in effective BER of all the signal combination techniques.

Spatial diversity [4] is a scheme in which more than one transmitter and/or receiver antennas are used to create multiple copies of a signal at the receiver. When the multiple paths taken by the signal are independent of each other, the copies arriving at the receiver at the same time will differ. The spacing between the antennas must be 0.38 times the wavelength of the carrier signal for the propagation paths to be independent of each other. An example of spatial diversity on the transmitter side of a communication system, is when a cellular phone is within range of two or more cell towers. Transmitter diversity is preferred over receiver diversity to keep mobile receivers light. The drawback of spatial diversity is that extra infrastructure

is needed for more antennas and receiver complexity.

Time Diversity [4] is a scheme in which a signal is transmitted more than once over the same medium. As in a Rayleigh fading environment, the characteristics of the channel vary randomly in a Rayleigh distribution, the copies of the same signal reaching the receiver at different times will differ. The receiver utilizes this difference to select the copy with the lowest distortion. The drawback of time diversity is that more time is spent in sending the same data, hence decreasing the overall data rate of the system and preventing it from being used in a static or slow-fading environment.

Frequency Diversity [4] is a scheme in which a signal is transmitted at different carrier frequencies over the same medium. The disadvantages of this method of diversity a that it cannot be used in a frequency-flat environment and a bandwidth is underutilized.

Polarization diversity [4] uses pairs of antennas with orthogonal polarizations to counteract signal fading caused by polarization mismatches due to reflections.

Pattern Diversity [4] uses multiple directional antennas with different radiation pattern to enhance the gain when compared to a single omni-directional antenna.

1.4 Alamouti Space-Time Coding

Alamouti space-time coding (STC) [1] is a diversity scheme in which space and time diversity are combined to improve the system performance. Space diversity is used to build a an $m \times n$ system, where m is the number of transmitter antennas, and n is the number of receiver antennas. Time diversity is used to transmit copies of the same symbol multiple times.

Two transmitter antennas and multiple receiver antennas are used in Alamouti STC.

Each receiver antenna is considered to be part of an independent 2×1 system. Each receiver processes the received symbols with the channel coefficients corresponding to it, and arrives at the same transmitted symbols. The computed transmitted symbols from different receivers can be used with the signal diversity combination techniques given in section 1.3.

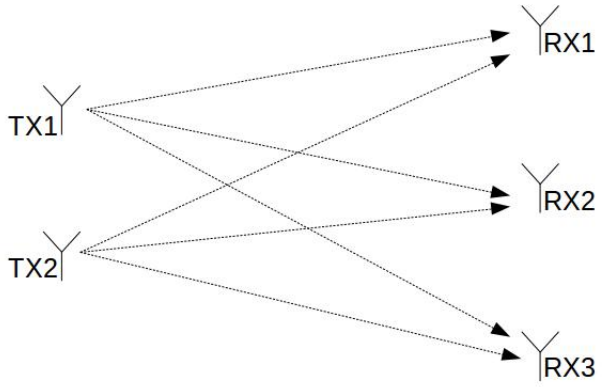

Figure 1.7: Setup of a 2x3 system that employs Alamouti space-time coding.

This thesis assumes a 2×1 Alamouti system. If two symbols s_1 and s_2 are to be transmitted, they are transmitted in the fashion given in Table 1.1.

	Antenna 1	**Antenna 2**
time 0	s_1	s_2
time 0 + T	$-s_2^*$	s_1^*

Table 1.1: Order of transmission of symbols in Alamouti STC [1]

The channel coefficients are assumed to be constant over two symbol periods. They are

$$h_1 = |h_1|e^{j\angle h_1}$$

and

$$h_2 = |h_2|e^{j\angle h_2}.$$

The received signals at a receiver during two symbol periods are given by

$$x_1 = h_1 s_1 + h_2 s_2 + n_1$$

and

$$x_2 = -h_1 s_2^* + h_2 s_1^* + n_2.$$

The relationship between the received and the transmitted symbol matrices is given by

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} s_1 & s_2 \\ -s_2^* & s_1^* \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} + \begin{bmatrix} n_1 \\ n_2 \end{bmatrix}.$$

Given that the nature of the noise matrix n in the above equation is random, the maximum likelihood detector in the receiver ignores the noise and solves for the transmitted symbols in the following manner:

$$\begin{bmatrix} x_1 \\ x_2^* \end{bmatrix} = \begin{bmatrix} h_1 & h_2 \\ h_2^* & -h_1^* \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix},$$

$$\begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} h_1 & h_2 \\ h_2^* & -h_1^* \end{bmatrix}^{-1} \begin{bmatrix} x_1 \\ x_2^* \end{bmatrix}$$

$$\begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = \frac{1}{|h_1|^2 + |h_2|^2} \begin{bmatrix} h_1^* & h_2 \\ h_2^* & -h_1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2^* \end{bmatrix},$$

$$s_1 = \frac{h_1^* x_1 + h_2 x_2^*}{|h_1|^2 + |h_2|^2}$$

and

$$s_2 = \frac{-h_1 x_2^* + h_2^* x_1}{|h_1|^2 + |h_2|^2}.$$

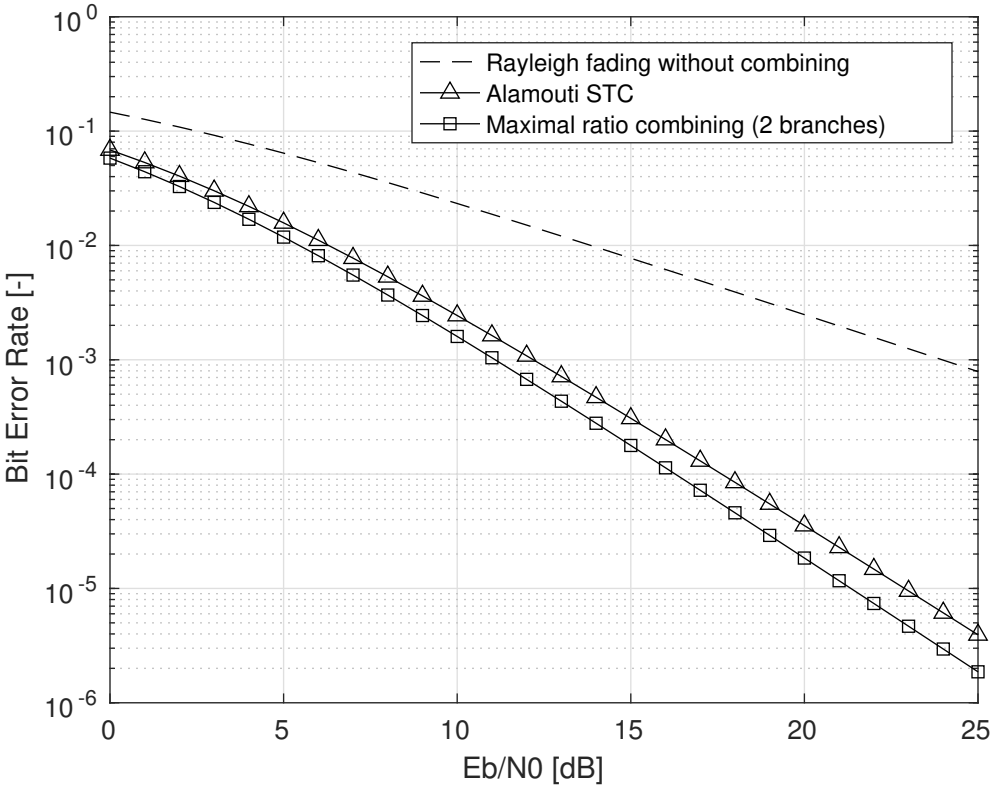

Figure 1.8: Bit-error rate performance of a QPSK constellation in Alamouti STC.

In spite of a BER performance that is worse than that of maximal-ratio combining, the advantage of Alamouti STC is that spatial and time diversity are used for better performance, and the net bit-error rate is decreased without decreasing the overall data rate of the communication system.

1.5 Blind Estimation of Alamouti STC

A system that employs Alamouti STC needs to have prior knowledge of the channel characteristics to deduce the transmitted symbols from the received symbols. The transmitter can send a training signal every few symbols so that the receiver can compute the channel characteristics. The disadvantages of this method are:

- As the training signal is sent in place of data, the net data rate decreases.

- If the channel characteristics change very fast when compared to the rate at which signals are transmitted, the values of the channel coefficients obtained by the receiver could be incorrect.

Blind estimation is a process through which the receiver estimates the transmitted signal from the received signal, without prior knowledge of the channel characteristics. Blind estimation can also be used to study the properties of the channel. A search of the literature reveals blind estimation techniques which can be categorized as follows:

- Higher order statistics

- Decision aided

- Time frame comparison

Three blind estimation methods are explained below.

1.5.1 Fourth-Order Statistics

Iglesias, Naya et al. [10] proposed a method of estimating the channel characteristics based on 4th-order cumulant matrices. The summary of the proposed method is given in Algorithm 1 which is at the end of Chapter 1.

1.5.2 Second-Order Statistics

Iglesias, Naya et al. [11] proposed a method of estimating the channel characteristics based on 2^{nd} order statistics. The summary of the proposed method is given below

[3, 9, 11].

The covariance matrix of the received symbols in Alamouti STC can be given by

$$C = E[XX^H],$$

$$C = (HS + n)(HS + n)^H$$

and

$$C = HR_sH^H + \sigma_n^2 I_2.$$

where

- $X = \begin{bmatrix} x_1 \\ x_2^* \end{bmatrix}$.
- $H = \begin{bmatrix} h_1 & h_2 \\ h_2^* & -h_1^* \end{bmatrix}$.
- $R_s = \begin{bmatrix} |s_1|^2 & 0 \\ 0 & |s_2|^2 \end{bmatrix}$.
- $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

Since H is orthogonal, we can rewrite the above equation as

$$C = H(R_s + \frac{\sigma_n^2}{||h||^2} I_2)H^H.$$

The channel matrix corresponds to the eigenvectors of C. C is diagonal when two consecutive transmitted symbols are radiated with the same power. Hence, C cannot undergo eigenvector decomposition for channel estimation. When the powers of two consecutive symbols are different,

$$E[|s_2^2|] = \gamma^2 E[|s_1^2|] = \gamma^2 \sigma_s^2.$$

$$C = \sigma_s^2 H \triangle_{SOS} H^H.$$

where

$$\triangle_{SOS} = \begin{bmatrix} 1+\sigma_h^2 & 0 \\ 0 & \gamma^2+\sigma_h^2 \end{bmatrix}$$

The channel matrix H can be obtained by applying eigenvalue decomposition to C. The eigenvalues of C are $1+\sigma_h^2$ and $\gamma^2+\sigma_h^2$.

The first eigenvector can be obtained by solving for V_1 in

$$(C - (1+\sigma_h^2)I)V_1 = 0.$$

The second eigenvector can be obtained by solving for V_2 in

$$(C - (\gamma^2+\sigma_h^2)I)V_2 = 0.$$

The channel matrix can be estimated by

$$H = \begin{bmatrix} V_1 & V_2 \end{bmatrix}.$$

1.5.3 Time-Frame Comparison

Zhou, Zhang and Wong [13] proposed a method of estimating the channel characteristics based on time-frame comparison.

During two Alamouti time-frames the transmitted symbols, received symbols and channel characteristics are assumed to be as follows:

	1^{st} **time-frame**	2^{nd} **time-frame**
Transmitted symbols	s_{1i}, s_{2i}	$s_{1(i+1)}, s_{2(i+1)}$
Received symbols	z_{1i}, z_{2i}	$z_{1(i+1)}, z_{2(i+1)}$
Channel characteristics	h_1, h_2	h_1, h_2

Table 1.2: Notations of terms in the Time-Frame Comparison method

Assuming that the power emitted by the transmitter antennas is 1, then

$$\begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} s_{1i}^* & -s_{2i} \\ s_{2i}^* & s_{1i} \end{bmatrix} \begin{bmatrix} z_{1i} \\ z_{2i} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} s_{1(i+1)}^* & -s_{2(i+1)} \\ s_{2(i+1)}^* & s_{1(i+1)} \end{bmatrix} \begin{bmatrix} z_{1(i+1)} \\ z_{2(i+1)} \end{bmatrix}$$

Simplifying this equation, we get

$$\begin{bmatrix} z_{1(i+1)} \\ z_{2(i+1)} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} s_{1(i+1)} & s_{2(i+1)} \\ -s_{2(i+1)}^* & s_{1(i+1)}^* \end{bmatrix} \begin{bmatrix} s_{1i}^* & -s_{2i} \\ s_{2i}^* & s_{1i} \end{bmatrix} \begin{bmatrix} z_{1i} \\ z_{2i} \end{bmatrix}$$

Let $a = \frac{1}{2}(s_{1(i+1)}s_{1i}^* + s_{2(i+1)}s_{2i}^*)$, $b = \frac{1}{2}(s_{2(i+1)}s_{1i} + s_{1(i+1)}s_{2i})$ from which we get,

$$\begin{bmatrix} z_{1(i+1)} \\ z_{2(i+1)}^* \end{bmatrix} = \begin{bmatrix} z_{1i} & z_{2i} \\ z_{2i}^* & -z_{1i}^* \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$$

$$Z_2 = Z_1 S_c$$

$$S_c = Z_1^{-1} Z_2$$

The elements of S_c are fed into a maximum likelihood detector (MLD) to obtain the transmitted symbols. For the MLD to obtain a unique solution, symbols from consecutive Alamouti time-frames need to be transmitted with different powers.

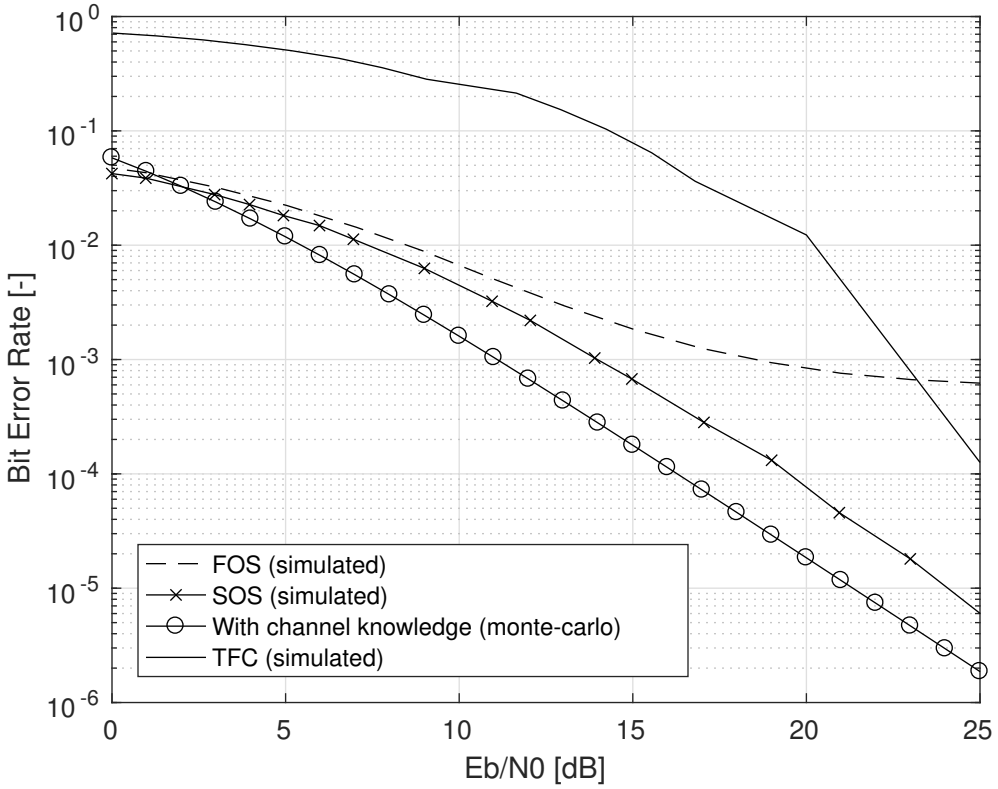

Figure 1.9: Comparison of bit-error rate performances of blind estimation approaches of a QPSK constellation in Alamouti STC, with prior knowledge of the channel coefficients.

Algorithm 1 Fourth-order statistics method

1: Note:

$$c_4(x_1, x_2, x_3, x_4) = E[x_1x_2x_3x_4] - E[x_1x_2]E[x_3x_4] - E[x_1x_3]E[x_2x_4] - E[x_1x_4]E[x_2x_3].$$

2: Compute

$$C[1,1]^{2,2} = c_4(x_2, x_2^*, x_1, x_1^*)$$

and

$$C[1,1]^{1,2} = c_4(x_1, x_2^*, x_1, x_1^*).$$

3: **if** $\frac{|C[1,1]^{2,2}|}{|C[1,1]^{1,2}|} < 1$ **then**

$$[k,l] = [1,1],$$
$$C[1,1]^{1,1} = c_4(x_1, x_1^*, x_1, x_1^*)$$

and

$$C[1,1]^{2,1} = (C[1,1]^{1,2})^*.$$

4: **else**

$$[k,l] = [1,2],$$
$$C[1,2]^{1,2} = c_4(x_1, x_2^*, x_1, x_2^*)$$
$$C[1,2]^{2,2} = c_4(x_2, x_2^*, x_1, x_2^*),$$
$$C[1,2]^{1,1} = (C[1,1]^{1,2}),$$

and

$$C[1,2]^{2,1} = (C[1,1]^{2,2}).$$

5: **end if**

6: Compute the eigenvalues and eigenvectors using

$$\lambda_i = \frac{t \pm \sqrt{t^2 + 4d}}{2}, i = 1, 2$$

where

$$t = C[k,l]^{1,1} + C[k,l]^{2,2}$$

and

$$d = C[k,l]^{2,1}C[k,l]^{1,2} - C[k,l]^{1,1}C[k,l]^{2,2}.$$

7: The corresponding eigenvectors are

$$u_i^{'} = \begin{bmatrix} \frac{C[k,l]^{1,2}}{\lambda_i - C[k,l]^{1,1}} \\ 1 \end{bmatrix}, u_i = \frac{u_i^{'}}{||u_i^{'}||}$$

8: The channel matrix is estimated by H=$[u_1\ u_2]$.

Chapter 2

New Method for Blind Estimation of Alamouti STC

The primary assumption made in this method is that the transmitter uses QPSK constellations with two different powers for consecutive symbols. An example of the constellations used is: Constellation 1: $\begin{Bmatrix} 1+1i & 1-1i & -1-1i & -1+1i \end{Bmatrix}$ and Constellation 2: $\begin{Bmatrix} 0.6+0.6i & 0.6-0.6i & -0.6-0.6i & -0.6+0.6i \end{Bmatrix}$.

If X is the matrix of the received symbols in the Alamouti scheme ,the covariance matrix of the received symbols is

$$C = E[XX^H]$$

and

$$C = HVH^H.$$

By expanding the above equation, we get

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} |h_1|^2 + r|h_2|^2 & (1-r)h_1h_2 \\ (1-r)(h_1h_2)^{'} & |h_2|^2 + r|h_1|^2 \end{bmatrix},$$

r being the ratio of the powers of the two constellations used. By solving the above expression, we get

$$|h_1| = \sqrt{\frac{C_{11} - rC_{22}}{1-r^2}},$$

$$|h_2| = \sqrt{\frac{C_{22} - rC_{11}}{1-r^2}}$$

and

$$\theta = \angle h_1 + \angle h_2 = \angle C_{12}.$$

Let α_1 and α_2 be the angles of s_1 and s_2 respectively, and ϕ_1 and ϕ_2 be the angles of h_1 and h_2 respectively, then

$$x_1 = h_1s_1 + \sqrt{r}h_2s_2 = |h_1s_1|e^{j(\phi_1+\alpha_1)} + \sqrt{r}|h_2s_2|e^{j(\phi_2+\alpha_2)}$$

and

$$x_2 = -\sqrt{r}h_1^{'}s_2 + h_2^{'}s_1 = -\sqrt{r}|h_1s_2|e^{j(-\phi_1+\alpha_2)} + |h_2s_1|e^{j(-\phi_2+\alpha_1)}.$$

By solving the above equations, we get

$$|h_1|x_1 + |h_2|e^{j\theta}x_2 = (|h_1|^2 + |h_2|^2)|s_1|e^{j(\alpha_1+\phi_1)},$$

$$|h_2|x_1 - |h_1|e^{j\theta}x_2 = (|h_1|^2 + |h_2|^2)\sqrt{r}|s_2|e^{j(\alpha_2+\phi_2)},$$

$$|h_2|x_2 + |h_1|e^{-j\theta}x_1 = (|h_1|^2 + |h_2|^2)|s_1|e^{j(\alpha_1-\phi_2)}$$

and

$$-|h_1|x_2 + |h_2|e^{-j\theta}x_1 = (|h_1|^2 + |h_2|^2)\sqrt{r}|s_2|e^{j(\alpha_2-\phi_1)}.$$

From which, we get

$$\theta_1 = \alpha_1 + \phi_1 = \angle(|h_1|x_1 + |h_2|e^{j\theta}x_2),$$

$$\theta_2 = \alpha_2 + \phi_2 = \angle(|h_2|x_1 - |h_1|e^{j\theta}x_2),$$

$$\theta_3 = \alpha_1 - \phi_2 = \angle(|h_2|x_2 + |h_1|e^{-j\theta}x_1)$$

and

$$\theta_4 = \alpha_2 - \phi_1 = \angle(-|h_1|x_2 + |h_2|e^{-j\theta}x_1).$$

The above expressions represent a system of linear equations with four variables $\alpha_1, \alpha_2, \phi_1$ and ϕ_2. Therefore, the phases of the symbols and the channel coefficients can be determined by solving

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \phi_1 \\ \phi_2 \end{bmatrix} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{bmatrix}$$

$$C_s\alpha = P.$$

The transmitted symbols can be recovered from the computed phases. But, the expression does not have a unique solution, because the coefficients matrix C_s is singular. Obtaining the transmitted symbols from the above system of equations is a possible goal for future research.

2.1 Algorithm to perform Partial Blind Estimation of Alamouti STC

The steps to be implemented to estimate the transmitted symbols are given in Algorithm 2.

2.2 Algorithm to Calculate the Angle of a Complex Number

In Algorithm 2, the arctangents of C_{12}, e_1, e_2, e_3 and e_4 need to be calculated to accomplish blind estimation. The algorithm used to approximate the arctangent needs to be suitable for fixed-point implementation. In [8], Lyons proposed an equation to approximate the arctangent of a complex number. It is given by

$$tan^{-1}(\frac{Q}{I}) \approx \frac{\frac{Q}{I}}{1 + 0.28125(\frac{Q}{I})^2},$$

where Q and I are the imaginary and real parts, respectively.

The error in the values computed using the above equation is shown in Figure 2.1.

Algorithm 2 Blind estimation of Alamouti STC

1: Approximate the covariance matrix of the received symbols using

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = E[xx^H].$$

2: Approximate the magnitudes of the channel coefficients using

$$|h_1| = \sqrt{\frac{C_{11} - r * C_{22}}{1 - r^2}}$$

and

$$|h_2| = \sqrt{\frac{C_{22} - r * C_{11}}{1 - r^2}}.$$

3: Estimate the sum of the phases of the channel coefficients using Algorithm 3.

4: Calculate the following values:

$$e_1 = |h_1|r_1 + |h_2|e^{j\theta}r_2,$$

$$e_2 = |h_2|r_1 - |h_1|e^{j\theta}r_2,$$

$$e_3 = |h_2|r_2 + |h_1|e^{-j\theta}r_1$$

and

$$e_4 = -|h_1|r_2 + |h_2|e^{-j\theta}r_1.$$

5: Estimate θ_1 using Algorithm 4.

6: Estimate θ_2 using Algorithm 5.

7: Estimate θ_3 using Algorithm 6.

8: Estimate θ_4 using Algorithm 7.

9: To get the phases of the transmitted symbols and the channel coefficients, solve

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \phi_1 \\ \phi_2 \end{bmatrix} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{bmatrix}.$$

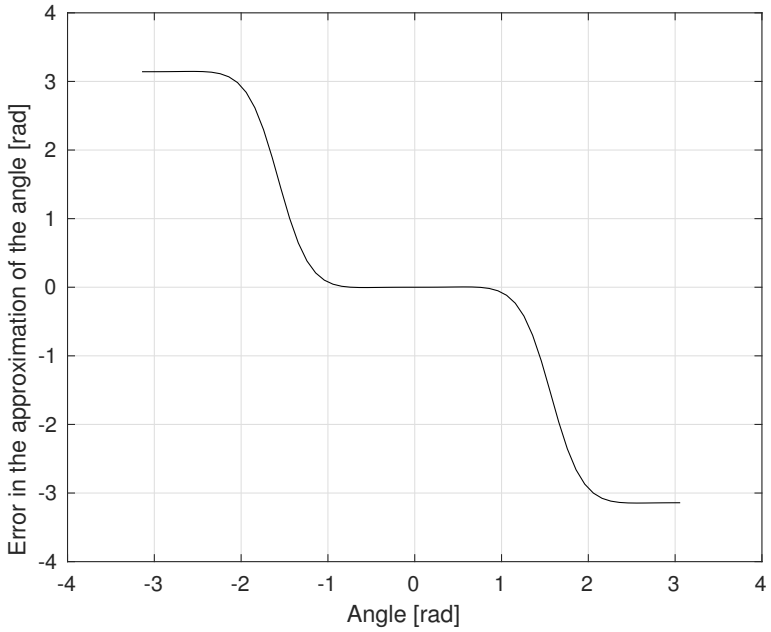

Figure 2.1: Plot of the error in the angle obtained from the equation proposed in [8].

Figure 2.1 shows that the above equation can be used satisfactorily only when the angle is in the range $[\frac{-\pi}{4}\ \frac{\pi}{4}]$. Angles outside the range $[\frac{-\pi}{4}\ \frac{\pi}{4}]$ can be estimated from the equations in table 2.1. These equations are implemented step by step in fixed-point in Algorithms 3, 4, 5, 6 and 7 to approximate θ, θ_1, θ_2, θ_3 and θ_4 respectively.

Quadrant	**Arctan Approximation** $(tan^{-1}(\frac{Q}{I}))$
1^{st}	$\frac{\frac{Q}{I}}{1+0.28125(\frac{Q}{I})^2}$
2^{nd}	$\frac{\pi}{2} - \frac{\frac{Q}{I}}{1+0.28125(\frac{Q}{I})^2}$
3^{rd}	$\frac{\pi}{2} - \frac{\frac{Q}{I}}{1+0.28125(\frac{Q}{I})^2}$
4^{th}	$\pi + \frac{\frac{Q}{I}}{1+0.28125(\frac{Q}{I})^2}$

Table 2.1: Approximation of arctan in all quadrants

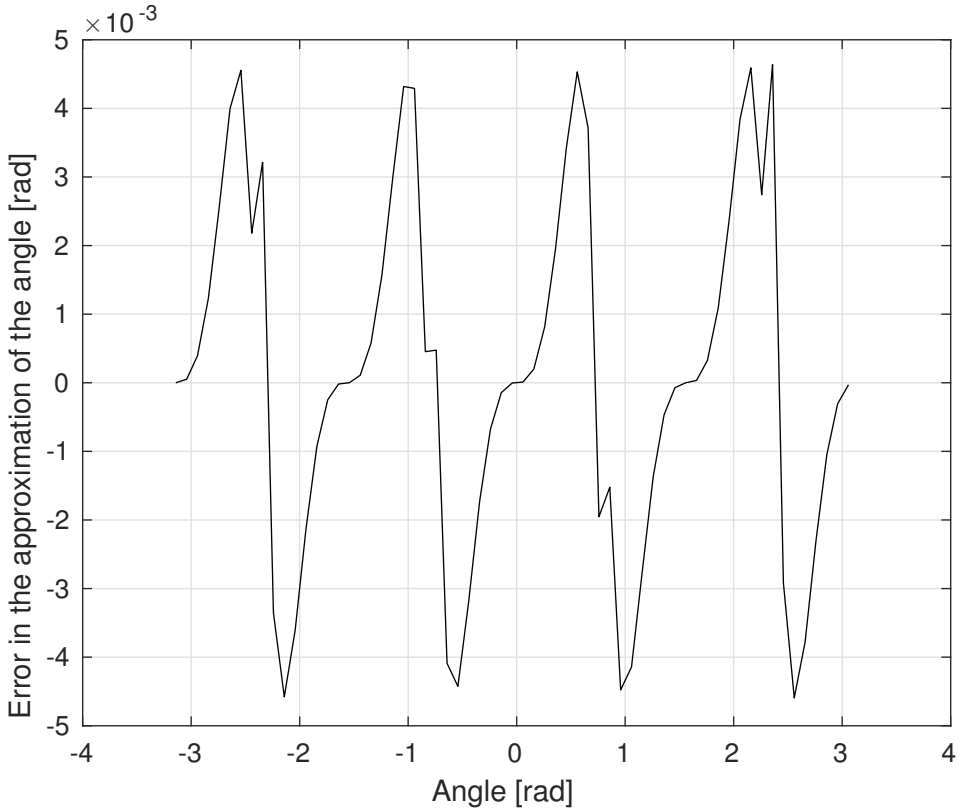

Figure 2.2: Plot of the error in the angle obtained from the equations proposed in table 2.1.

Algorithm 3 Estimation of the sum of the angles

$$a = \frac{imaginary(C_{12})}{real(C_{12})}$$

if $a < 0$ **then** $a_1 = -a$.

  if $a_1 > 1$ **then**

$$a_2 = \frac{1}{a_1}$$

$$p = \frac{\pi}{2} - \frac{a_2}{1 + 0.28125a_2^2}$$

  else if $a_1 < 1$ **then**

$$a_2 = a_1$$

$$p = \frac{a_2}{1 + 0.28125a_2^2}.$$

  end if

  if $real(C_{12}) > 0$ & $imaginary(C_{12}) < 0$ **then**

$$\theta = -p.$$

  else if $real(C_{12}) < 0$ & $imaginary(C_{12}) > 0$ **then**

$$\theta = \pi - p.$$

  else if $a_1 = 1$, $real(C_{12}) > 0$ & $imaginary(C_{12}) < 0$ **then**

$$\theta = \frac{-\pi}{4}.$$

  else if $a_1 = 1$, $real(C_{12}) < 0$ & $imaginary(C_{12}) > 0$ **then**

$$\theta = \frac{3\pi}{4}.$$

  end if

else if $a > 0$ **then**, $a_1 = a$.

  if $a_1 > 1$ **then**

$$a_2 = \frac{1}{a_1}$$

$$p = \frac{\pi}{2} - \frac{a_2}{1 + 0.28125a_2^2}.$$

  else if $a_1 < 1$ **then**

$$a_2 = a_1$$

$$p = \frac{a_2}{1 + 0.28125a_2^2}.$$

  end if

Algorithm 3 (continued)

if $real(C_{12}) > 0$ & $imaginary(C_{12}) > 0$ **then**

$$\theta = p.$$

.

else if $real(C_{12}) < 0$ & $imaginary(C_{12}) < 0$ **then** $\theta = -\pi + p$.

else if $a_1 = 1$, $real(C_{12}) > 0$ & $imaginary(C_{12}) > 0$ **then**

$$\theta = \frac{\pi}{4}.$$

else if $a_1 = 1$, $real(C_{12}) < 0$ & $imaginary(C_{12}) < 0$ **then**

$$\theta = \frac{-3\pi}{4}.$$

end if

else if $a = 0$ **then**

if $real(C_{12}) > 0$ & $imaginary(C_{12}) = 0$ **then**

$$\theta = 0.$$

else if $real(C_{12}) < 0$ & $imaginary(C_{12}) = 0$ **then**

$$\theta = \pi.$$

else if $real(C_{12}) = 0$ & $imaginary(C_{12}) > 0$ **then**

$$\theta = \frac{\pi}{2}.$$

else if $real(C_{12}) = 0$ & $imaginary(C_{12}) < 0$ **then**

$$\theta = \frac{-\pi}{2}.$$

end if

end if

Algorithm 4 Estimation of θ_1

$$a = \frac{imaginary(e_1)}{real(e_1)}.$$

if $a < 0$ **then** $a_1 = -a$.
  if $a_1 > 1$ **then**

$$a_2 = \frac{1}{a_1}$$

$$p = \frac{\pi}{2} - \frac{a_2}{1 + 0.28125a_2^2}$$

  else if $a_1 < 1$ **then**

$$a_2 = a_1$$

$$p = \frac{a_2}{1 + 0.28125a_2^2}.$$

  end if
  if $real(e_1) > 0$ & $imaginary(e_1) < 0$ **then**

$$\theta_1 = -p.$$

  else if $real(e_1) < 0$ & $imaginary(e_1) > 0$ **then**

$$\theta_1 = \pi - p.$$

  else if $a_1 = 1$, $real(e_1) > 0$ & $imaginary(e_1) < 0$ **then**

$$\theta_1 = \frac{-\pi}{4}.$$

  else if $a_1 = 1$, $real(e_1) < 0$ & $imaginary(e_1) > 0$ **then**

$$\theta_1 = \frac{3\pi}{4}.$$

  end if
else if $a > 0$ **then**, $a_1 = a$.
  if $a_1 > 1$ **then**

$$a_2 = \frac{1}{a_1}.$$

$$p = \frac{\pi}{2} - \frac{a_2}{1 + 0.28125a_2^2}.$$

  else if $a_1 < 1$ **then**

$$a_2 = a_1.$$

$$p = \frac{a_2}{1 + 0.28125a_2^2}.$$

  end if

Algorithm 4 (continued)

if $real(e_1) > 0$ & $imaginary(e_1) > 0$ **then**

$$\theta_1 = p.$$

else if $real(e_1) < 0$ & $imaginary(e_1) < 0$ **then** $\theta_1 = -\pi + p$.

else if $a_1 = 1$, $real(e_1) > 0$ & $imaginary(e_1) > 0$ **then**

$$\theta_1 = \frac{\pi}{4}.$$

else if $a_1 = 1$, $real(e_1) < 0$ & $imaginary(e_1) < 0$ **then**

$$\theta_1 = \frac{-3\pi}{4}.$$

end if

else if $a = 0$ **then**

if $real(e_1) > 0$ & $imaginary(e_1) = 0$ **then**

$$\theta_1 = 0.$$

else if $real(e_1) < 0$ & $imaginary(e_1) = 0$ **then**

$$\theta_1 = \pi.$$

else if $real(e_1) = 0$ & $imaginary(e_1) > 0$ **then**

$$\theta_1 = \frac{\pi}{2}.$$

else if $real(e_1) = 0$ & $imaginary(e_1) < 0$ **then**

$$\theta_1 = \frac{-\pi}{2}.$$

end if

end if

Algorithm 5 Estimation of θ_2

$$a = \frac{imaginary(e_2)}{real(e_2)}.$$

if $a < 0$ **then** $a_1 = -a$.
  if $a_1 > 1$ **then**

$$a_2 = \frac{1}{a_1}$$

$$p = \frac{\pi}{2} - \frac{a_2}{1 + 0.28125a_2^2}$$

  else if $a_1 < 1$ **then**

$$a_2 = a_1$$

$$p = \frac{a_2}{1 + 0.28125a_2^2}.$$

  end if
  if $real(e_2) > 0$ & $imaginary(e_2) < 0$ **then**

$$\theta_2 = -p.$$

.
  else if $real(e_2) < 0$ & $imaginary(e_2) > 0$ **then**

$$\theta_2 = \pi - p.$$

  else if $a_1 = 1$, $real(e_2) > 0$ & $imaginary(e_2) < 0$ **then**

$$\theta_2 = \frac{-\pi}{4}.$$

  else if $a_1 = 1$, $real(e_2) < 0$ & $imaginary(e_2) > 0$ **then**

$$\theta_2 = \frac{3\pi}{4}.$$

  end if
else if $a > 0$ **then**, $a_1 = a$.
  if $a_1 > 1$ **then**

$$a_2 = \frac{1}{a_1}$$

$$p = \frac{\pi}{2} - \frac{a_2}{1 + 0.28125a_2^2}$$

  else if $a_1 < 1$ **then**

$$a_2 = a_1$$

$$p = \frac{a_2}{1 + 0.28125a_2^2}.$$

  end if

Algorithm 5 (continued)

if $real(e_2) > 0$ & $imaginary(e_2) > 0$ **then**

$$\theta_2 = p.$$

else if $real(e_2) < 0$ & $imaginary(e_2) < 0$ **then** $\theta_2 = -\pi + p$.

else if $a_1 = 1$, $real(e_2) > 0$ & $imaginary(e_2) > 0$ **then**

$$\theta_2 = \frac{\pi}{4}.$$

else if $a_1 = 1$, $real(e_2) < 0$ & $imaginary(e_2) < 0$ **then**

$$\theta_2 = \frac{-3\pi}{4}.$$

end if

else if $a = 0$ **then**

if $real(e_2) > 0$ & $imaginary(e_2) = 0$ **then**

$$\theta_2 = 0.$$

else if $real(e_2) < 0$ & $imaginary(e_2) = 0$ **then**

$$\theta_2 = \pi.$$

else if $real(e_2) = 0$ & $imaginary(e_2) > 0$ **then**

$$\theta_2 = \frac{\pi}{2}.$$

else if $real(e_2) = 0$ & $imaginary(e_2) < 0$ **then**

$$\theta_2 = \frac{-\pi}{2}.$$

end if

end if

Algorithm 6 Estimation of θ_3

$$a = \frac{imaginary(e_3)}{real(e_3)}.$$

if $a < 0$ **then** $a_1 = -a$.
  if $a_1 > 1$ **then**

$$a_2 = \frac{1}{a_1}$$

$$p = \frac{\pi}{2} - \frac{a_2}{1 + 0.28125a_2^2}.$$

  else if $a_1 < 1$ **then**

$$a_2 = a_1$$

$$p = \frac{a_2}{1 + 0.28125a_2^2}.$$

  end if
  if $real(e_3) > 0$ & $imaginary(e_3) < 0$ **then**

$$\theta = -p.$$

  else if $real(e_3) < 0$ & $imaginary(e_3) > 0$ **then**

$$\theta = \pi - p.$$

  else if $a_1 = 1$, $real(e_3) > 0$ & $imaginary(e_3) < 0$ **then**

$$\theta = \frac{-\pi}{4}.$$

  else if $a_1 = 1$, $real(e_3) < 0$ & $imaginary(e_3) > 0$ **then**

$$\theta = \frac{3\pi}{4}.$$

  end if
else if $a > 0$ **then**, $a_1 = a$.
  if $a_1 > 1$ **then**

$$a_2 = \frac{1}{a_1}$$

$$p = \frac{\pi}{2} - \frac{a_2}{1 + 0.28125a_2^2}.$$

  else if $a_1 < 1$ **then**

$$a_2 = a_1$$

$$p = \frac{a_2}{1 + 0.28125a_2^2}.$$

  end if

Algorithm 6 (continued)

if $real(e_3) > 0$ & $imaginary(e_3) > 0$ **then**

$$\theta = p.$$

else if $real(e_3) < 0$ & $imaginary(e_3) < 0$ **then** $\theta = -\pi + p$.

else if $a_1 = 1$, $real(e_3) > 0$ & $imaginary(e_3) > 0$ **then**

$$\theta = \frac{\pi}{4}.$$

else if $a_1 = 1$, $real(e_3) < 0$ & $imaginary(e_3) < 0$ **then**

$$\theta = \frac{-3\pi}{4}.$$

end if

else if $a = 0$ **then**

if $real(e_3) > 0$ & $imaginary(e_3) = 0$ **then**

$$\theta = 0.$$

else if $real(e_3) < 0$ & $imaginary(e_3) = 0$ **then**

$$\theta = \pi.$$

else if $real(e_3) = 0$ & $imaginary(e_3) > 0$ **then**

$$\theta = \frac{\pi}{2}.$$

else if $real(e_3) = 0$ & $imaginary(e_3) < 0$ **then**

$$\theta = \frac{-\pi}{2}.$$

end if

end if

Algorithm 7 Estimation of θ_4

$$a = \frac{imaginary(e_4)}{real(e_4)}.$$

if $a < 0$ **then** $a_1 = -a$.
  if $a_1 > 1$ **then**

$$a_2 = \frac{1}{a_1}$$

$$p = \frac{\pi}{2} - \frac{a_2}{1 + 0.28125a_2^2}.$$

  else if $a_1 < 1$ **then**

$$a_2 = a_1$$

$$p = \frac{a_2}{1 + 0.28125a_2^2}.$$

  end if
  if $real(e_4) > 0$ & $imaginary(e_4) < 0$ **then**

$$\theta = -p.$$

  else if $real(e_4) < 0$ & $imaginary(e_4) > 0$ **then**

$$\theta = \pi - p..$$

  else if $a_1 = 1$, $real(e_4) > 0$ & $imaginary(e_4) < 0$ **then**

$$\theta = \frac{-\pi}{4}.$$

  else if $a_1 = 1$, $real(e_4) < 0$ & $imaginary(e_4) > 0$ **then**

$$\theta = \frac{3\pi}{4}.$$

  end if
else if $a > 0$ **then**, $a_1 = a$.
  if $a_1 > 1$ **then**

$$a_2 = \frac{1}{a_1}$$

$$p = \frac{\pi}{2} - \frac{a_2}{1 + 0.28125a_2^2}.$$

  else if $a_1 < 1$ **then**

$$a_2 = a_1$$

$$p = \frac{a_2}{1 + 0.28125a_2^2}.$$

  end if

Algorithm 7 (continued)

if $real(e_4) > 0$ & $imaginary(e_4) > 0$ **then**

$$\theta = p.$$

else if $real(e_4) < 0$ & $imaginary(e_4) < 0$ **then** $\theta = -\pi + p$.

else if $a_1 = 1$, $real(e_4) > 0$ & $imaginary(e_4) > 0$ **then**

$$\theta = \frac{\pi}{4}.$$

else if $a_1 = 1$, $real(e_4) < 0$ & $imaginary(e_4) < 0$ **then**

$$\theta = \frac{-3\pi}{4}.$$

end if

else if $a = 0$ **then**

if $real(e_4) > 0$ & $imaginary(e_4) = 0$ **then**

$$\theta = 0.$$

else if $real(e_4) < 0$ & $imaginary(e_4) = 0$ **then**

$$\theta = \pi.$$

else if $real(e_4) = 0$ & $imaginary(e_4) > 0$ **then**

$$\theta = \frac{\pi}{2}.$$

else if $real(e_4) = 0$ & $imaginary(e_4) < 0$ **then**

$$\theta = \frac{-\pi}{2}.$$

end if

end if

Chapter 3

Implementation of Partial Blind Estimation of Alamouti STC

The new method mentioned in chapter 2 cannot yet be used for complete blind estimation due to the coefficient matrix in step 9 being singular. Hence, the method is implemented only until the estimation of the magnitudes and the sum of the phases of the channel coefficients. The block diagram of the implemented system is given in figure 3.1. The process of implementation is divided into three parts:

1. **Covariance Matrix Computation**: This set of blocks processes the received symbols to compute the elements of the covariance matrix, and sends them to the expected value computation set of blocks.

   The covariance matrix of the received symbols is

$$XX^H = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \begin{bmatrix} x_1^* & x_2^* \end{bmatrix}$$

and

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} |x_1|^2 & x_1 x_2^* \\ x_1^* x_2 & |x_2|^2 \end{bmatrix}.$$

Therefore,

$$C_{11} = real(x_1)^2 + imag(x_1)^2,$$

$$C_{12} = (real(x_1)real(x_2)+imag(x_1)imag(x_2))+i(real(x_2)imag(x_1)-real(x_1)imag(x_2)),$$

$$C_{21} = (real(x_1)real(x_2)+imag(x_1)imag(x_2))-i(real(x_2)imag(x_1)-real(x_1)imag(x_2)) = C_{12}^*,$$

$$C_{22} = real(x_2)^2 + imag(x_2)^2,$$

$$real(C_{12}) = real(C_{21}) = C_{12r} = real(x_1)real(x_2) + imag(x_1)imag(x_2)$$

and

$$imag(C_{12}) = -imag(C_{21}) = C_{12i} = real(x_2)imag(x_1) - real(x_1)imag(x_2).$$

If the matrix of the real and imaginary parts of the received symbols is

$$\mathbf{X}_c = \begin{bmatrix} real(x_1) \\ imag(x_1) \\ real(x_2) \\ imag(x_2) \end{bmatrix},$$

the relationships between the outputs and the inputs are

$$C_{11} = \mathbf{X}_c^T \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \mathbf{X}_c,$$

$$C_{22} = \mathbf{X}_c^T \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{X}_c,$$

$$real(C_{12}) = \mathbf{X}_c^T \begin{bmatrix} 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0.5 \\ 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \end{bmatrix} \mathbf{X}_c$$

and

$$imag(C_{12}) = \mathbf{X}_c^T \begin{bmatrix} 0 & 0 & 0 & -0.5 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0.5 & 0 & 0 \\ -0.5 & 0 & 0 & 0 \end{bmatrix} \mathbf{X}_c.$$

The hardware implementation of this part can be done in parallel and serial. In serial implementation which was performed in software on the ARM core, input memory is required for all the received symbols, and a serial to parallel converter is required at the output for all the computed values. In parallel implementation which was performed on hardware on the FPGA core on the Zedboard, no memory is required for the received symbols or computed values. The block diagram of this part is given in figure 3.2.

2. **Expected Value Computation**: This set of blocks aggregates the elements of the covariance matrix, and sends them to the channel coefficient computation set of blocks. If the number of symbols considered for blind estimation is N, the relationships between the outputs and the inputs are

$$E[C_{11}] = \frac{1}{N}\sum_{1}^{N} C_{11},$$

$$E[C_{22}] = \frac{1}{N}\sum_{1}^{N} C_{22},$$

$$E[real(C_{12})] = \frac{1}{N}\sum_{1}^{N} real(C_{12})$$

and

$$E[imag(C_{12})] = \frac{1}{N}\sum_{1}^{N} imag(C_{12}).$$

Serial implementation which was performed in hardware on the FPGA core on the Zedboard, requires input memory. Parallel implementation which was performed in software on the ARM core, requires input memory and a serial to parallel converter. The block diagrams of the two kinds of implementations of this part are given in figures 3.3 and 3.4.

3. **Channel Coefficient Computation**: This set of blocks receives the elements of the expected value of the covariance matrix, and computes the magnitudes and the sum of the phases of the channel coefficients. The block diagram of this part is given in figure 3.4. If the ratio of the powers of two consecutive transmitted symbols is r, the relationships between the outputs and the inputs are

$$|h_1| = \sqrt{\frac{E[C_{11}] - r * E[C_{22}]}{1 - r^2}},$$

$$|h_2| = \sqrt{\frac{E[C_{22}] - r * E[C_{11}]}{1 - r^2}}$$

and

$$\angle h_1 h_2 = tan^{-1}\left(\frac{E[imag(C_{12})]}{E[real(C_{12})]}\right).$$

Serial implementation which was performed in hardware on the FPGA core on the Zedboard as well as in software on the ARM core, requires no memory. Parallel implementation cannot be performed for this part because only one set of input values exists for a set of received symbols. The arctan function can be implemented using an approximate equation proposed by Lyons [8],as given in Algorithms 3,4,5,6 and 7 as implemented in software on the ARM core, or using a CORDIC arctan function [12] as implemented in hardware on the FPGA core on the Zedboard.

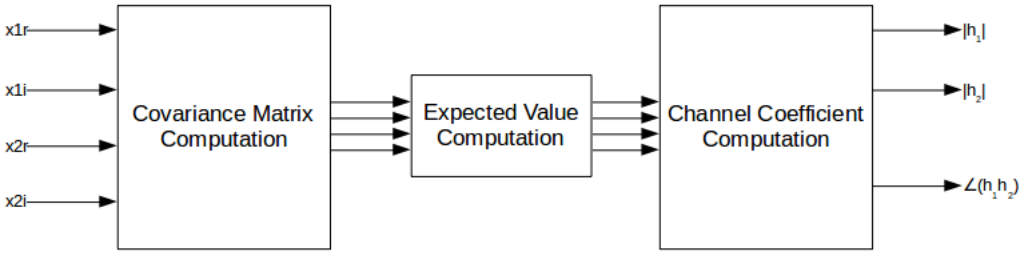

Figure 3.1: Block Diagram of partial blind estimation of Alamouti STC

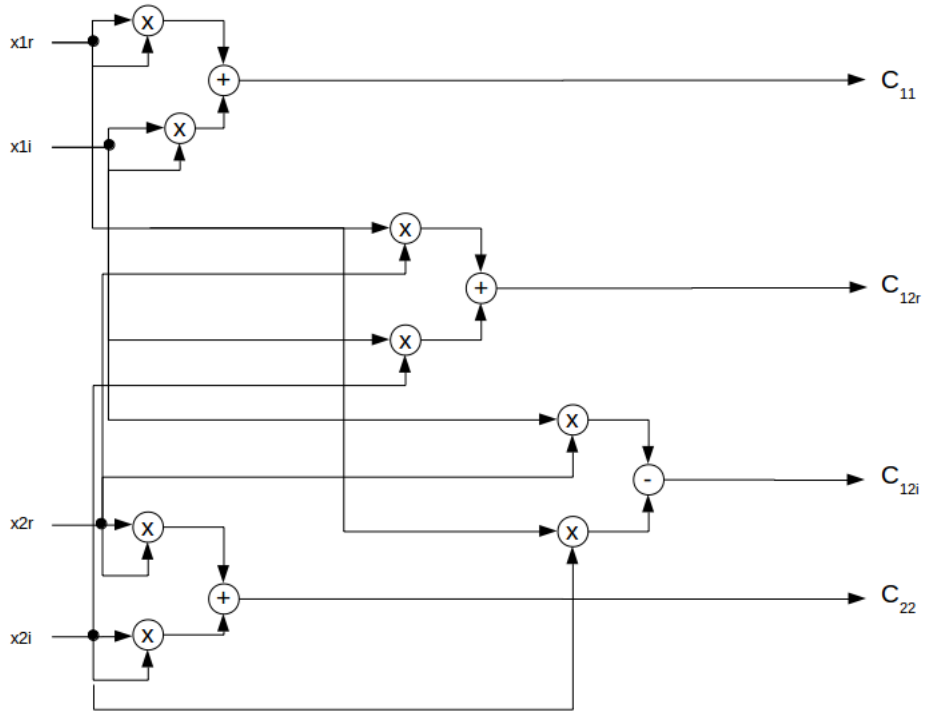

Figure 3.2: Block diagram of covariance matrix computation

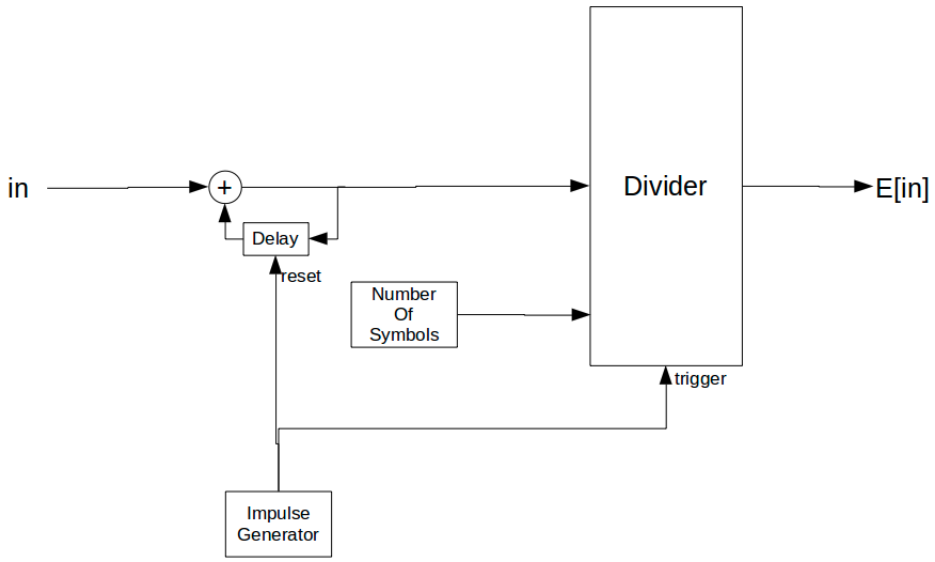

Figure 3.3: Block diagram of expected value computation in serial implementation.

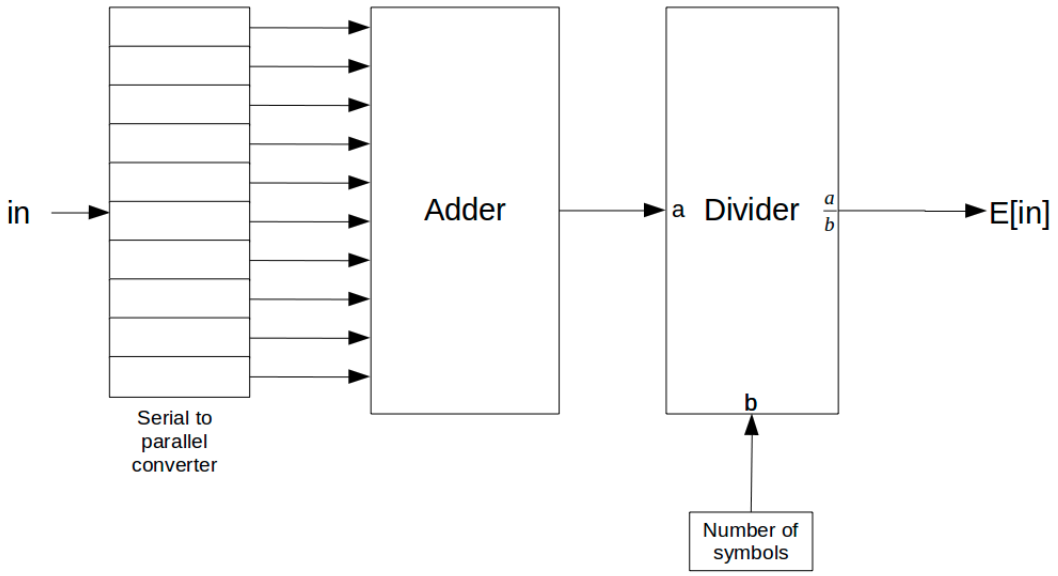

Figure 3.4: Block diagram of expected value computation in parallel implementation.

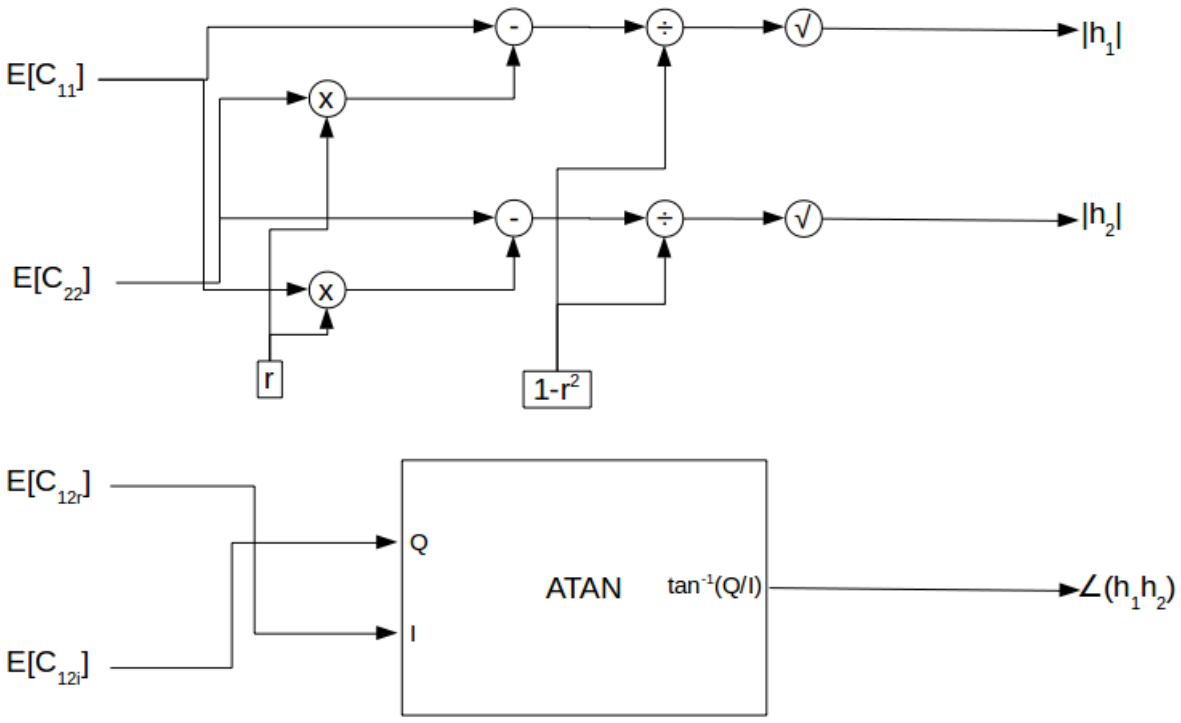

Figure 3.5: Block diagram of channel estimation

3.1 Simulation Results of Proposed Algorithm

The three parts of the algorithm proposed above were simulated multiple times in MATLAB. The MATLAB codes are given in Appendix A. The relationships studied between the parameters recorded from the simulation are:

- *Estimated* $|h_1|$ versus *Actual* $|h_1|$ (Figure 3.6).

- *Estimated* $|h_2|$ versus *Actual* $|h_2|$ (Figure 3.7).

- *Estimated* $\angle h_1 + \angle h_2$ versus *Actual* $\angle h_1 + \angle h_2$ (Figure 3.8).

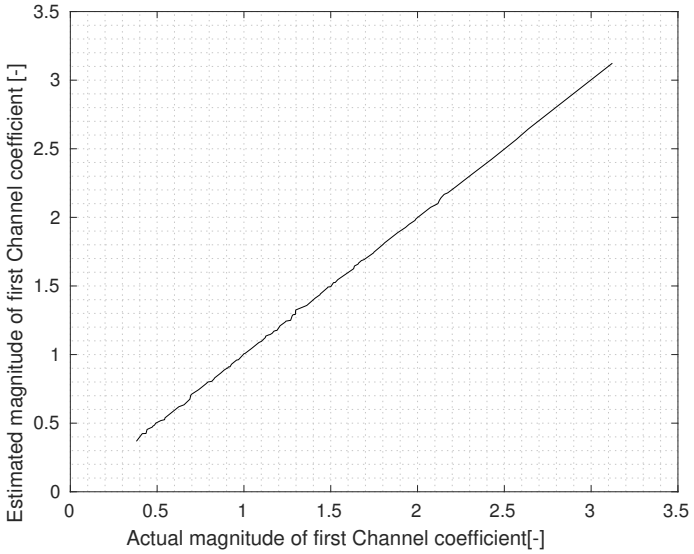

Figure 3.6: Comparison of the actual values and the estimated values of the magnitude of the first channel coefficient obtained from simulation in MATLAB.

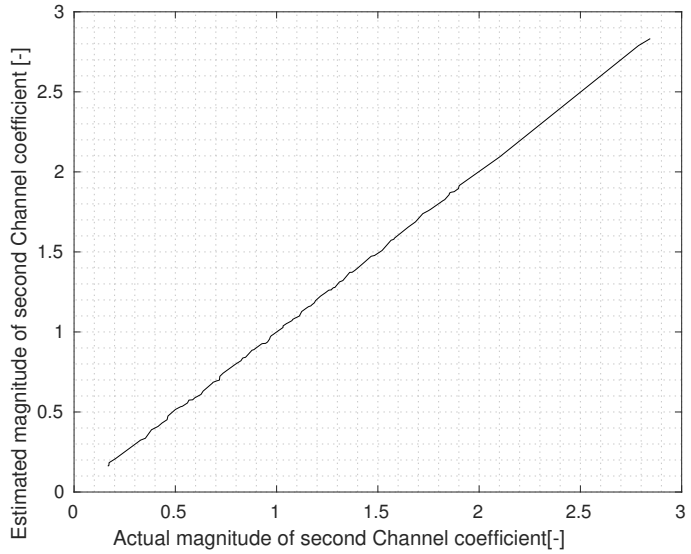

Figure 3.7: Comparison of the actual values and the estimated values of the magnitude of the second channel coefficient obtained from simulation in MATLAB.

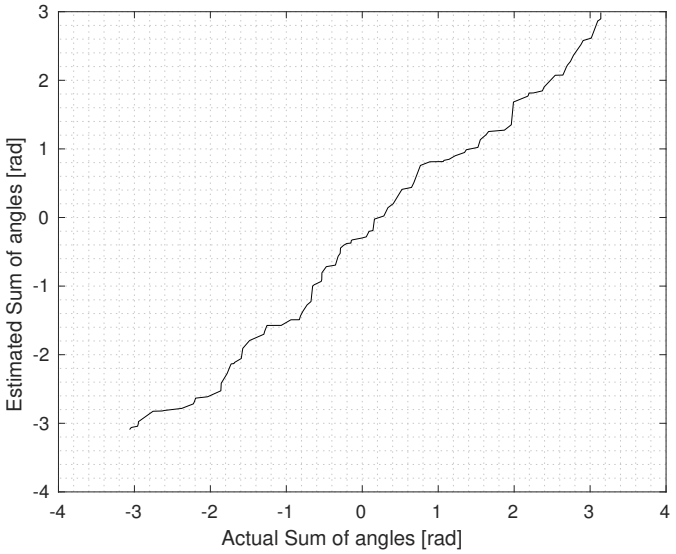

Figure 3.8: Comparison of the actual values and the estimated values of the sum of the angles of the channel coefficients obtained from simulation in MATLAB,

The observations from the data collected from the MATLAB simulation are:

1. The maximum error in the values estimated for the magnitudes of the first and second channel coefficients are 3.71% and 10.27% respectively.

2. The minimum error in the values estimated for the magnitudes of the first and second channel coefficients are 0.0043% and 0.0066% respectively.

3. The maximum and minimum errors in the values estimated for the sum of the angles of the channel coefficients are 0.441 radians and 1.7867×10^{-4} radians respectively.

4. The error in the estimated values of the magnitude of the second channel coefficient is higher than that of the first.

5. The error in the estimated values of the sum of the angles of the channel coefficients is more random and less predictable than that of the magnitudes of the coefficients.

From the above observations, we can conclude that due to the error in the estimated values being low, the proposed method can be used for hardware implementation of partial blind estimation of Alamouti STC.

Chapter 4

Software Implementation on ARM

This chapter explains and studies the performance of the software implementation of partial blind estimation. The covariance matrix computation, expected value computation and channel coefficient computation functions were compiled and executed in C on the ARM-core on Zedboard. The C codes for the functions are given in Appendix B. The C program utilizing the 3 functions was compiled, and the executable file was executed on the ARM core running Xubuntu Linux Software implementation of the algorithm running on hardware is preferred over hardware implementation when there is a need to frequently modify the execution of the implemented functions, for e.g. changing the number of symbols considered for computation of the channel coefficients, changing the ratio of the powers of the two constellations used, etc.

The technical specifications of the ARM core on Zedboard are:

- Number of cores: 2

- Name of ARM processor: ARM ® Cortex™-A9 MPCore™

- Maximum clock speed: 667 MHz

The number of the cores on the Zedboard is 2. As the Linux kernel runs on one core, only one core was used for the execution for the implementation and testing of the partial blind estimation algorithm. The values of r, h_1 and h_2 used for the execution were 0.6, $0.9\angle 0.9$ and $0.6\angle 0.832$ respectively. The received symbols were generated using an Alamouti model coded in C. The received symbols generated were noise-free, to isolate the effect of the algorithm on the accuracy of the results. Tests were run multiple times on the hardware to understand the nature of the results of the algorithm executed on the C core.

4.1 Performance of Partial Blind Estimation on ARM

The relationships, between the parameters recorded from the ARM core, studied are:

- $|h_1|$ versus *number of received symbols* (Figure 4.1).
- $|h_2|$ versus *number of received symbols* (Figure 4.2).
- $\angle h_1 + \angle h_2$ versus *number of received symbols* (Figure 4.3).
- *Processing time* versus *number of received symbols* (Figure 4.4).
- *Symbol-error rate* versus *Number of symbols used for estimation*(Figure 4.5).
- *Bit-error rate* versus *Signal to noise ratio*(Figure 4.6).

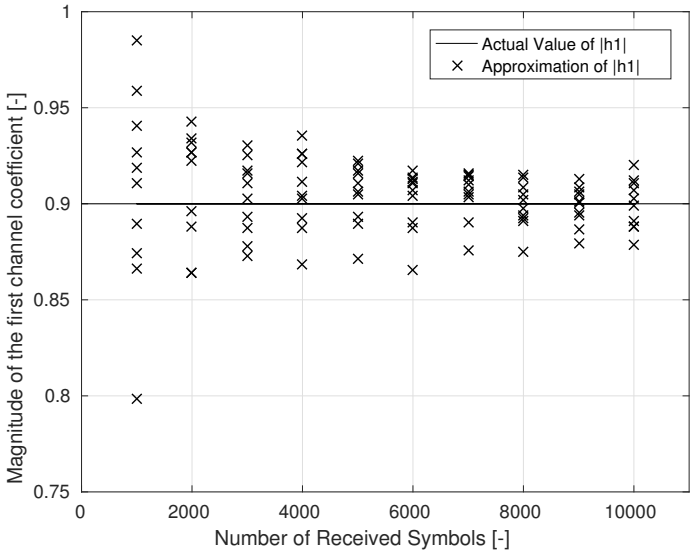

Figure 4.1: Comparison of the actual value of the magnitude of the first channel coefficient and the values obtained from execution of partial blind estimation of the channel characteristics of a system employing Alamouti STC with a QPSK constellation, on the ARM core.

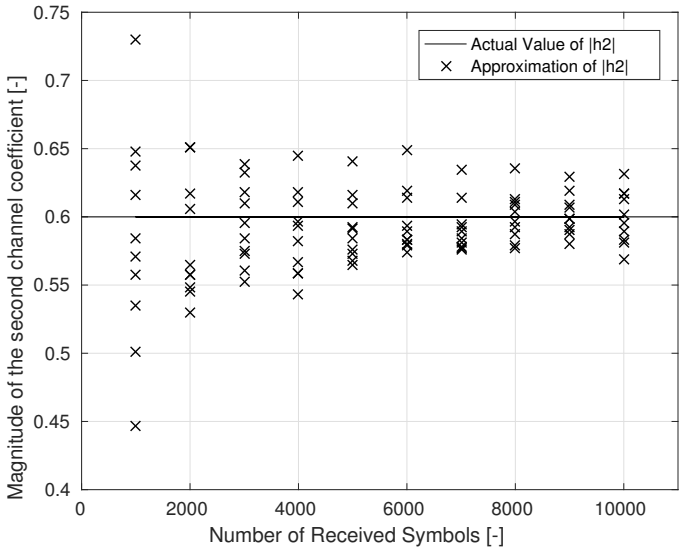

Figure 4.2: Comparison of the actual value of the magnitude of the second channel coefficient and the values obtained from execution of partial blind estimation of the channel characteristics of a system employing Alamouti STC with a QPSK constellation, on the ARM core.

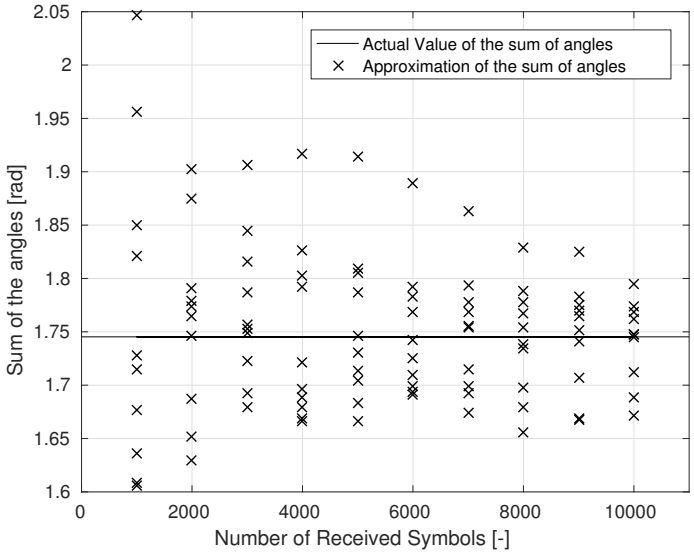

Figure 4.3: Comparison of the actual value of the sum of the phases of the channel coefficients and the values obtained from execution of partial blind estimation of the channel characteristics of a system employing Alamouti STC with a QPSK constellation, on the ARM core.

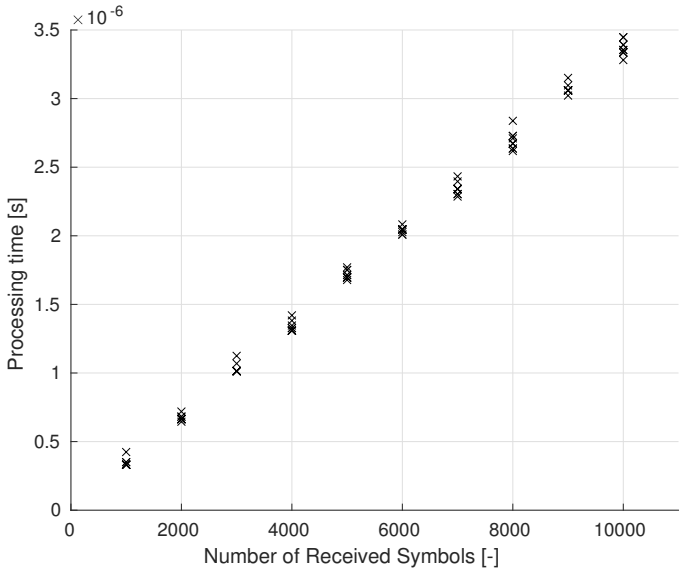

Figure 4.4: Time taken by the ARM core to process the partial blind estimation algorithm for the channel coefficients of Alamouti STC with a QPSK constellation, using different lengths of the stream of received symbols.

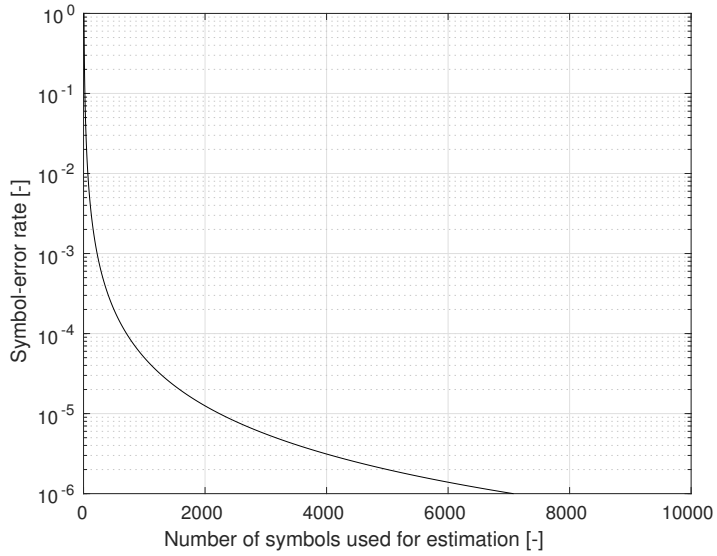

Figure 4.5: Change in the Symbol-error rate as the number of symbols considered changes, in a system with a QPSK constellation utilizing blind estimation of Alamouti STC without noise and with prior knowledge of the phases of the channel coefficients on the ARM core.

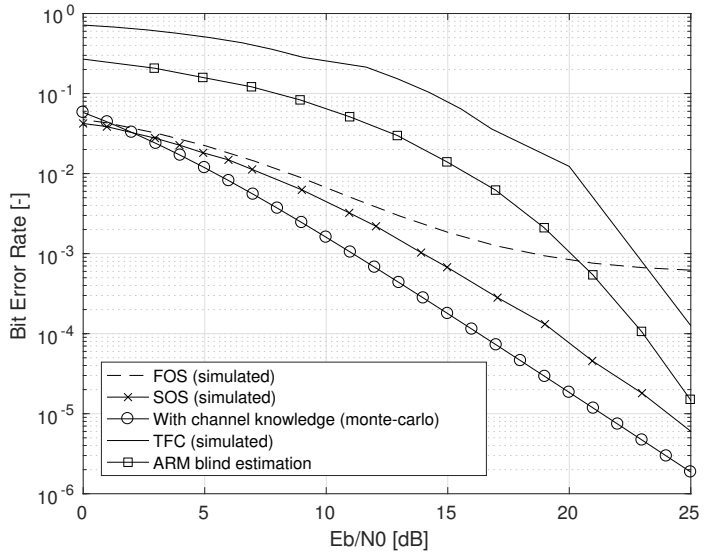

Figure 4.6: Comparison of the bit-error rate performances of the partial blind estimation algorithm implemented on the ARM core with prior knowledge of the phases of the channel coefficients utilizing 10000 symbols, the fourth-order statistics method, the second-order statistics method, and the time-frame comparison method, executed in a system utilizing Alamouti STC with a QPSK constellation.

The observations from the data collected from the ARM implementation are:

1. Using more than 8000 symbols for computation produces a negligible increase in the accuracy of the computed data.

2. The time taken by the ARM core to process the partial blind estimation algorithm is directly proportional to the number of symbols processed.

3. The average processing time per symbol in an ARM core on Zedboard for the partial blind estimation algorithm is 3.389 ns.

4. The maximum error in the values estimated for the magnitudes of the first and second channel coefficients from 10000 symbols are 2.39% and 5.19% respectively.

5. The minimum error in the values estimated for the magnitudes of the first and second channel coefficients from 10000 symbols are 0.13% and 0.3% respectively.

6. The maximum and minimum errors in the values estimated for the sum of the angles of the channel coefficients are 0.07 radians and 6.71×10^{-4} radians respectively.

7. The error in the estimated values of the sum of the angles of the channel coefficients is more random and less predictable than that of the magnitudes of the coefficients.

4.2 Conclusion

The processing time of 3.389 ns per symbol limits the data rate to 590.145 megabits per second. This could act as a bottleneck in the whole communication system. Yet,

a constant processing time per symbol is useful in estimating the delay introduced in the system and making appropriate design changes. As the processing time increases with the number of symbols considered, and beyond 8000 symbols there is a negligible increase in performance, using only 8000 symbols to perform partial blind estimation is a good balance between processing delay and performance. Despite the limitation introduced in the data rate of the communication system, execution of the partial blind estimation algorithm on the ARM core could be instrumental in rapidly modifying the parameters used in the algorithm, either to achieve the desired performance or adapt to changes in the channel coefficients and transmitter characteristics. On the ARM core, the error in the computed values of $|h_2|$ is greater than that of $|h_1|$. This can be addressed by adjusting the parameters used in the algorithm during the execution to achieve the least error possible.

Chapter 5

Hardware Implementation on FPGA

This chapter explains and studies the performance of the software implementation of partial blind estimation. The partial blind estimation algorithm were implemented in simulink using the *Xilinx System Generator* blockset. The VHDL codes for the 3 functions proposed in chapter 3 are given in Appendix C. The Xilinx system generator block diagram was optimized in three different ways while generating the IP core to be executed in Vivado Design Suite:

- Power optimization to consume as less power as possible.
- Area optimization to utilize as less physical resources as possible.
- Runtime optimization to execute the algorithm as quickly as possible.

The resources available in the FPGA on Zedboard are given in Table 5.1.

The clock speed, processing delay and resource utilization of the FPGA during hardware implementation of the three optimized states are given in Table 5.2.

	Number of available resources [-]
Lookup table (LUT)	53200
Lookup table RAM (LUTRAM)	17400
Flip-flops (LUT)	106400
Block RAM [%]	140
DSP Slices (DSP)	220
Input-output ports (IO)	200
Global Buffer (BUFG)	32
Mixed-Mode Clock Manager (MMCM)	4

Table 5.1: Clock speed, processing delay and resource utilization of the FPGA during hardware implementation of the three optimized states

	Power-optimized	Area-optimized	Runtime-optimized
Clock Speed [MHz]	100	100	100
Delay [clock cycles]	53	53	53
LUT [%]	28.90	28.59	28.67
LUTRAM [%]	1.41	1.34	1.33
FF [%]	7.94	7.94	8.10
BRAM [%]	2.50	2.50	2.50
DSP [%]	16.82	16.82	16.82
IO [%]	0.50	0.50	0.50
BUFG [%]	12.50	12.50	12.50
MMCM [%]	25.00	25.00	25.00

Table 5.2: Clock speed, processing delay and resource utilization of the FPGA during hardware implementation of the three optimized states

Hardware implementation is preferred over software implementation when a system is to be deployed in the field, maximum performance is desired and there is no need to change the nature of the execution of the algorithm.

The partial blind estimation algorithm implemented on FPGA was a part of a simulated communication system designed in MATLAB. The values of r, h_1 and h_2 used for the execution were 0.6, $0.9\angle 0.23$ and $0.6\angle 0.16$ respectively. The system was simulated in a noise-free environment to isolate the effect of hardware implementation of the algorithm on the observed variables. Unlike the software implementation

on ARM, the execution was performed only once per optimization state, but for a longer time period because the system parameters have to be set while generating the hardware module, and every set of symbols considered for computation acts as an independent execution of the algorithm. The FPGA was programmed to consider 10000 symbols i.e. 5000 Alamouti time-frames for partial blind estimation at a time.

5.1 Performance of Partial Blind Estimation on FPGA

The relationships, between the parameters recorded from the FPGA implementation, studied are:

- $|h_1|$ *computed from a runtime-optimized state* versus *Number of received symbols* (Figure 5.1).

- $|h_2|$ *computed from a runtime-optimized state* versus *Number of received symbols* (Figure 5.2).

- $\angle h_1 + \angle h_2$ *computed from a runtime-optimized state* versus *Number of received symbols* (Figure 5.3).

- $|h_1|$ *computed from a area-optimized state* versus *Number of received symbols* (Figure 5.4).

- $|h_2|$ *computed from a area-optimized state* versus *Number of received symbols* (Figure 5.5).

- $\angle h_1 + \angle h_2$ *computed from a area-optimized state* versus *Number of received*

symbols (Figure 5.6).

- $|h_1|$ *computed from a power-optimized state* versus *Number of received symbols* (Figure 5.7).

- $|h_2|$ *computed from a power-optimized state* versus *Number of received symbols* (Figure 5.8).

- $\angle h_1 + \angle h_2$ *computed from a power-optimized state* versus *Number of received symbols* (Figure 5.9).

- *Bit-error rate* versus *Signal to noise ratio* (Figure 5.10).

The performance of the partial blind estimation algorithm on FPGA is shown in figures 5.1 to 5.10. The time delay in the partial blind estimation algorithm estimated on hardware is 50 clock cycles.

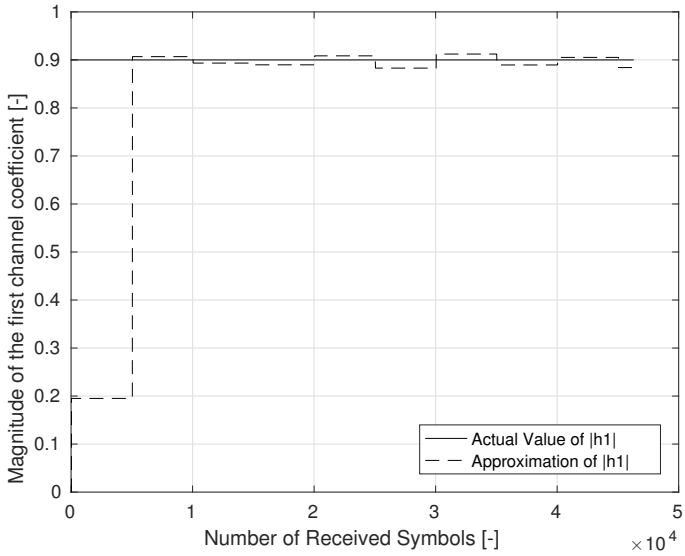

Figure 5.1: Comparison of the actual value of the magnitude of the first channel coefficient and the values obtained from execution of the partial blind-estimation algorithm of the channel coefficients of a system utilizing Alamouti STC with a QPSK constellation, on the FPGA running in a runtime-optimized state.

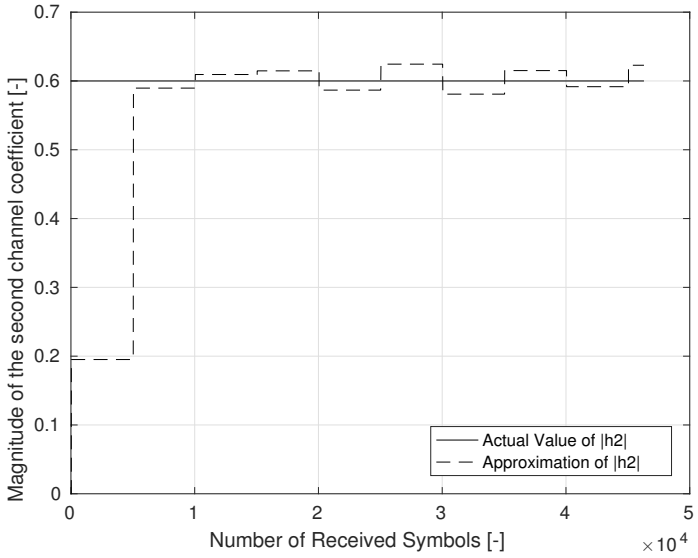

Figure 5.2: Comparison of the actual value of the magnitude of the second channel coefficient and the values obtained from execution of the partial blind-estimation algorithm of the channel coefficients of a system utilizing Alamouti STC with a QPSK constellation, on the FPGA running in a runtime-optimized state.

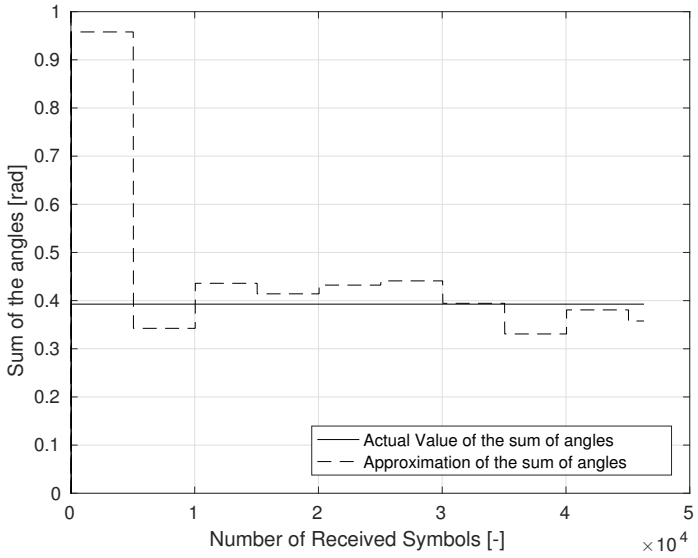

Figure 5.3: Comparison of the actual value of the sum of the phases of the channel coefficients and the values obtained from execution of the partial blind-estimation algorithm of the channel coefficients of a system utilizing Alamouti STC with a QPSK constellation, on the FPGA running in a runtime-optimized state.

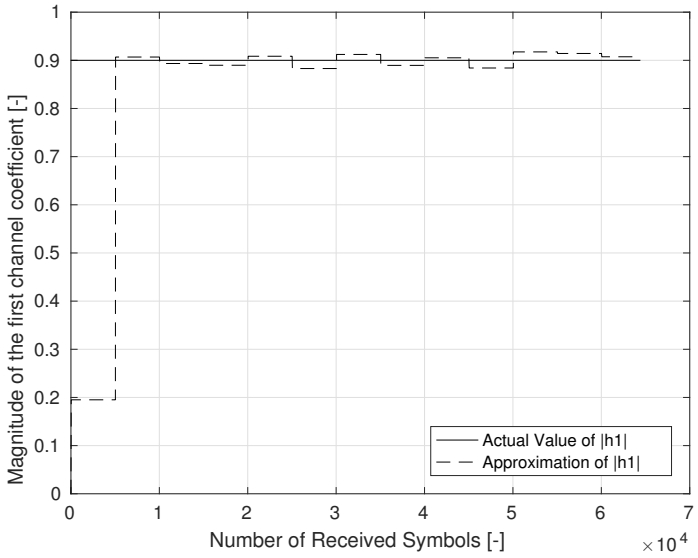

Figure 5.4: Comparison of the actual value of the magnitude of the first channel coefficient and the values obtained from execution of the partial blind-estimation algorithm of the channel coefficients of a system utilizing Alamouti STC with a QPSK constellation, on the FPGA running in an area-optimized state.

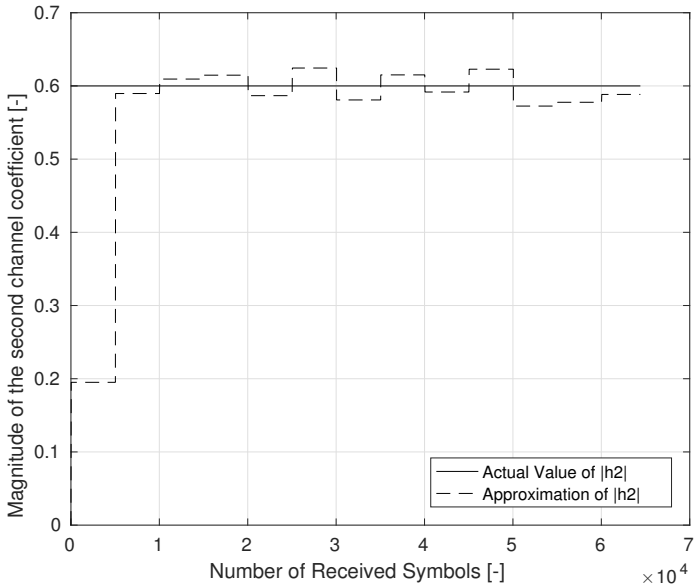

Figure 5.5: Comparison of the actual value of the magnitude of the second channel coefficient and the values obtained from execution of the partial blind-estimation algorithm of the channel coefficients of a system utilizing Alamouti STC with a QPSK constellation, on the FPGA running in an area-optimized state.

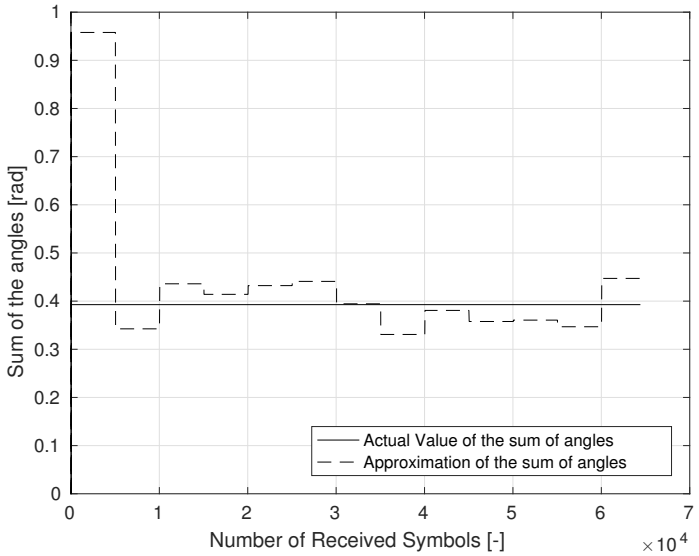

Figure 5.6: Comparison of the actual value of the sum of the phases of the channel coefficients and the values obtained from execution of the partial blind-estimation algorithm of the channel coefficients of a system utilizing Alamouti STC with a QPSK constellation, on the FPGA running in an area-optimized state.

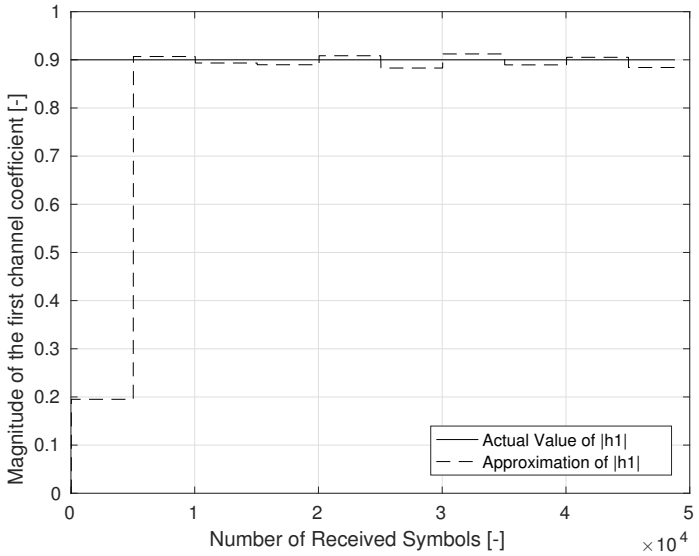

Figure 5.7: Comparison of the actual value of the magnitude of the first channel coefficient and the values obtained from execution of the partial blind-estimation algorithm of the channel coefficients of a system utilizing Alamouti STC with a QPSK constellation, on the FPGA running in a power-optimized state.

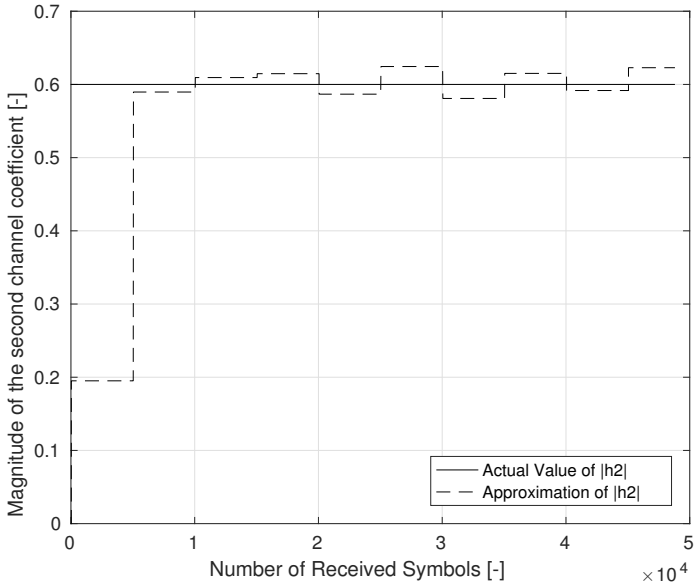

Figure 5.8: Comparison of the actual value of the magnitude of the second channel coefficient and the values obtained from execution of the partial blind-estimation algorithm of the channel coefficients of a system utilizing Alamouti STC with a QPSK constellation, on the FPGA running in a power-optimized state.

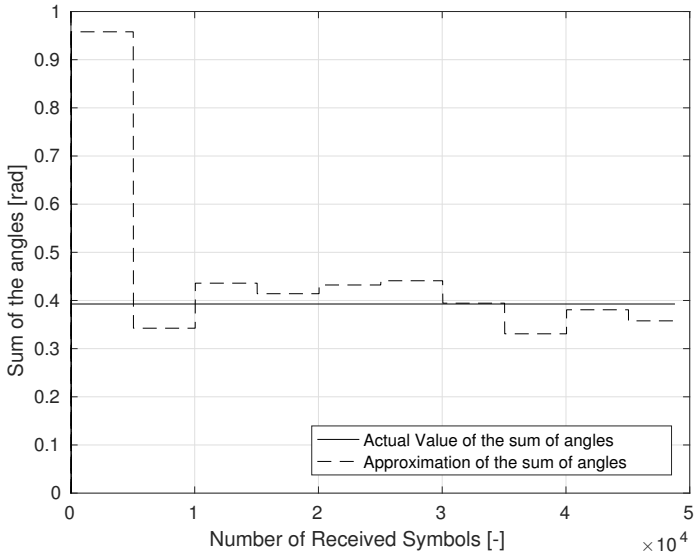

Figure 5.9: Comparison of the actual value of the sum of the phases of the channel coefficients and the values obtained from execution of the partial blind-estimation algorithm of the channel coefficients of a system utilizing Alamouti STC with a QPSK constellation, on the FPGA running in a power-optimized state.

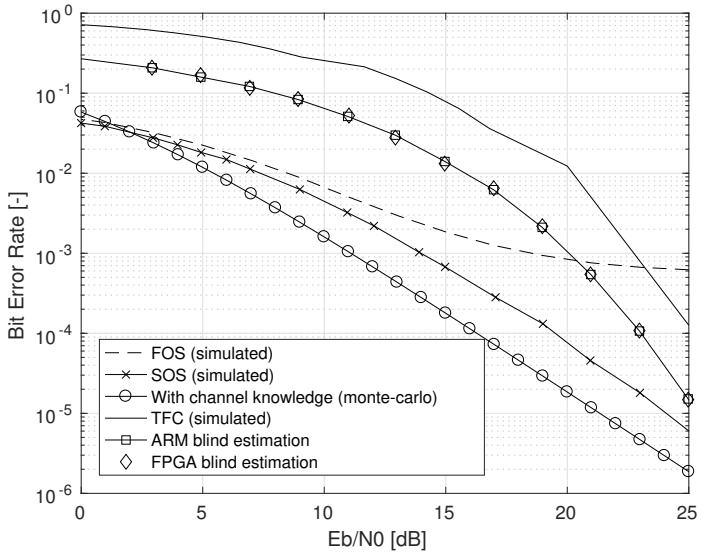

Figure 5.10: Comparison of the bit-error rate performances of the partial blind estimation algorithm implemented on the FPGA with prior knowledge of the phases of the channel coefficients utilizing 10000 symbols, the fourth-order statistics method, the second-order statistics method, and the time-frame comparison method.

The observations from the data collected from the FPGA implementation are:

1. There is no difference between the values computed from the FPGA in area-optimized, power-optimized and runtime-optimized states.

2. During the computation of the channel coefficients from the first set of received symbols, the values output by the FPGA are incorrect.

3. The value output by the FPGA remains constant for the period of the number of symbols considered, and changes every time a new set of received symbols is processed.

4. The error in the values computed from 10000 received symbols is the same as that obtained from the execution on the ARM core.

5. The number of clock cycles taken by the FPGA to process the partial blind estimation algorithm is 53.

6. The maximum error in the values estimated for the magnitudes of the first and second channel coefficients are 2.78% and 6.67% respectively.

7. The minimum error in the values estimated for the magnitudes of the first and second channel coefficients are 1.11% and 1.67% respectively.

8. The maximum and minimum errors in the values estimated for the sum of the angles of the channel coefficients are 0.441 radians and 0.05 radians respectively.

9. The error in the estimated values of the sum of the angles of the channel coefficients is more random and less predictable than that of the magnitudes of the coefficients.

5.2 Conclusion

Due to the absence of a difference between the values computed in the three separate optimized states, any suitable state can be used without any compromise in the accuracy in the output values. An FPGA-based partial blind estimation system is more suitable for a communication system with frequent changes in the channel characteristics, due to the output of the FPGA remaining constant for the duration of the symbols considered and changing at a regular rate. The partial blind estimation algorithm implemented on FPGA introduces a maximum data rate in the communication system. In the case of the implementation performed in the previous section, the clock rate and the processing delay being 100 MHz and 53 clock cycles for 10000 symbols respectively, the maximum data rate introduced is 37.736 gigabits per second. Similar to the ARM implementation the errors in the outputs differ between values. The error in the computed values of $|h_2|$ is greater than that of $|h_1|$. Unlike the case of the ARM, the parameters of the algorithm cannot be modified for the an improvement in the performance, during the execution.

Chapter 6

Conclusion

Blind estimation plays a key role in improving the performance communication systems. It increases the system's resilience to changes in the channel characteristics, and does not introduce an overhead to the data. For a blind estimation algorithm to be used by a communication system, the following qualities are preferred:

- Has a low rate of errors.
- Does not affect the performance of the overall system much.
- Does not decrease the performance capabilities of the system.

The algorithm proposed in chapter 2 is hypothesized to be able to perform complete blind estimation in a system utilizing Alamouti STC. But, it is incomplete due to the

coefficient matrix in step 9 of Algorithm 2 being singular and hence, the equation

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \phi_1 \\ \phi_2 \end{bmatrix} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{bmatrix}$$

being unsolvable. Therefore, Algorithm 2 was simulated and implemented on hardware only up to step 3. Some common observations from the data collected from the simulations and hardware implementations were:

- The mean error in the values computed for the magnitude of the second channel coefficient is higher than that of the first channel coefficient.

- The error in the estimated values of the sum of the angles of the channel coefficients is more random and less predictable than that of the magnitudes of the coefficients.

- The minimum delay introduced by the partial blind estimation algorithm in the communication system, is the duration of the number of symbols considered for computation.

The above common observations can be said to be the drawbacks of the implemented algorithm. All of these drawbacks need to be addressed by future research. Besides the drawbacks in the partial blind estimation module that are introduced by the algorithm that is implemented, the performance is affected by the method of implementation too. Both of the methods of hardware implementation i.e. on the ARM core and on the FPGA, have their place in the design and deployment of communication systems. Table 6.1 is a comparison between the two methods, which can be used when deciding which one to use.

ARM Core Implementation	FPGA Implementation
•Simple implementation	•Complex implementation
•Quick implementation	•Implementation takes time
•Can adapt algorithm to changes in system	•Cannot adapt algorithm to changes in system
•Can change parameters during execution	•Cannot change parameters during execution
•Suitable for algorithm testing	•Suitable for field-deployment
•High delay	•Low delay
•Low data rate	•High data rate

Table 6.1: Comparison of ARM core and FPGA implementations of the partial blind estimation of Alamouti STC algorithm.

The implementation performed in this thesis is of a very basic nature. There is a very large scope for improvement in the field of blind estimation of Alamouti STC. Algorithms to accomplish complete blind estimation in a system implementing Alamouti STC exist [5, 6, 10, 11, 13], but none of them have been implemented in a functioning communication system. This could be due to increased complexity, poor performance or too much overhead introduced in the proposed algorithms. Accomplishment of hardware implementation of complete blind estimation of Alamouti STC, will also motivate further research into blind estimation in conventional and MIMO communication systems.

Even though this thesis accomplished hardware implementation of a partial blind estimation algorithm, future research projects need to look into many questions that this piece of work was not able to answer, which are:

- Can the equation proposed in step 9 of Algorithm 2 be solved, even in an approximate manner? If yes, how?

- Can an algorithm that accomplishes complete blind estimation of Alamouti STC be implemented on hardware? If yes, how?

- How can the low data rate and high delay caused by the ARM implementation

be addressed and improved?

- How can a blind estimation algorithm be implemented in FPGAs while being able to adapt the algorithm and its parameters during its operation?

- Is there a method that can implement the blind estimation algorithm partly on the ARM and partly on the FPGA to improve the performance of the proposed algorithm when compared to the two individual methods?

Appendix A

MATLAB Code for Partial Blind Estimation

A.1 Expected Value of Covariance Matrix Computation

```
%% Covariance Matrix Computation
% Input: Real and imaginary parts of the recevied symbols,
% Output: C11, Real part of C12, Imaginary Part of C21, C22

function [c11,c12r,c12i,c22]=c_comp(x1r,x1i,x2r,x2i)

n=numel(x1r);
c11=0;
c12r=0;
c12i=0;
```

```
c22=0;

for j=1:1:n

% Computing C11 from the Received Symbols
c11=c11 + ((x1r(j)*x1r(j) + x1i(j)*x1i(j))/n);

% Computing the real part of C12 from the Received Symbols
c12r=c12r + ((x1r(j)*x2r(j) + x1i(j)*x2i(j))/n);%sum(x(1,:)
    .*conj(x(2,:)));

% Computing the imaginary part of C12 from the Received
    Symbols
c12i=c12i + ((x1i(j)*x2r(j) - x1r(j)*x2i(j))/n);%sum(x(1,:)
    .*conj(x(2,:)));

% Computing C22 from the Received Symbols
c22=c22 + ((x2r(j)*x2r(j) + x2i(j)*x2i(j))/n);

end

end
```

A.2 Channel Coefficient Computation

```
%% Channel Coefficient Computation
% Input: C11, Real part of C12, Imaginary Part of C21, C22
% Output: Magnitude of the first Channel Coefficient,
```

```
    Magnitude of the
%           Second Channel Coefficient, Sum of the angles of
    the  Channel
%           Coefficients

function [ah1,ah2,as_angh]=h_comp(c11,c12r,c12i,c22)

% Ratio of the Powers radiated by the two antennas
r=0.6;

% Computing the Magnitude of Channel Coefficient h1
ah1=sqrt(((c11-r*c22))/(1-(r^2)));

% Computing the Magnitude of Channel Coefficient h2
ah2=sqrt(((c22-r*c11))/(1-(r^2)));

%% Computing the sum of the angles of the channel
    characteristics
as_angh=0;
p=0;
a=0;
a1=0;
a2=0;

a=c12i/c12r;

if a<0
     a1=-a;
```

```
    if a1>1
        a2=1/a1;
        p=(pi/2)-(a2./(1+0.28125*a2.*a2));

        if c12r>0 & c12i <0
            as_angh=-p;
         elseif c12r<0 & c12i >0
            as_angh=pi-p;
        end
    elseif a1<1
        a2=a1;
        p=a2./(1+0.28125*a2.*a2);

        if c12r>0 & c12i <0
            as_angh=-p;
        elseif c12r<0 & c12i >0
            as_angh=pi-p;
        end
    elseif a1==1
        if c12r>0 & c12i <0
            as_angh=-pi/4;
        elseif c12r<0 & c12i >0
            as_angh=3*pi/4;
        end
    end
elseif a>0
    a1=a;
    if a1>1
```

```
        a2=1/a1;
        p=(pi/2)-(a2./(1+0.28125*a2.*a2));

        if c12r>0 & c12i >0
            as_angh=p;
        elseif c12r<0 & c12i <0
            as_angh=-pi+p;
        end
    elseif a1<1
        a2=a1;
        p=a2./(1+0.28125*a2.*a2);

        if c12r>0 & c12i >0
            as_angh=p;
        elseif c12r<0 & c12i <0
            as_angh=-pi+p;
        end
    elseif a1==1
        if c12r>0 & c12i >0
            as_angh=pi/4;
        elseif c12r<0 & c12i <0
            as_angh=-3*pi/4;
        end
    end
elseif a==0
            if c12r>0 & c12i==0
            as_angh=0;
        elseif c12r<0 & c12i==0
```

```
            as_angh=pi;
            end
else
            if c12i>0 & c12r==0
            as_angh=pi/2;
        elseif c12i<0 & c12r==0
            as_angh=-pi/2;
            end
end

end
```

Appendix B

C Code for Hardware Implementation of Partial Blind Estimation

B.1 Expected Value of Covariance Matrix Computation

```
/* Include Files */
#include "rt_nonfinite.h"
#include "c_comp.h"
#include "h_comp.h"
#include "r_gen.h"

/* Function Definitions */
```

```
/*
 * Arguments    : const emxArray_real32_T *x1r
 *                const emxArray_real32_T *x1i
 *                const emxArray_real32_T *x2r
 *                const emxArray_real32_T *x2i
 *                float *c11
 *                float *c12r
 *                float *c12i
 *                float *c22
 * Return Type  : void
 */
void c_comp(const emxArray_real32_T *x1r, const
    emxArray_real32_T *x1i,
                const emxArray_real32_T *x2r, const
                   emxArray_real32_T *x2i,
                float *c11, float *c12r, float *c12i, float
                   *c22)
{
  int fcnOutput;
  int j;
  fcnOutput = x1r->size[1];
  *c11 = 0.0F;
  *c12r = 0.0F;
  *c12i = 0.0F;
  *c22 = 0.0F;
  for (j = 0; j < (int)(float)x1r->size[1]; j++) {
    *c11 += (x1r->data[(int)(1.0F + (float)j) - 1] * x1r->
       data[(int)(1.0F +
```

```
            (float)j) - 1] + x1i->data[(int)(1.0F + (float
                )j) - 1] * x1i->
            data[(int)(1.0F + (float)j) - 1]) / (float)
               fcnOutput;

/*  Computing C11 from the Received Symbols */
*c12r += (x1r->data[(int)(1.0F + (float)j) - 1] * x2r->
   data[(int)(1.0F +
            (float)j) - 1] + x1i->data[(int)(1.0F + (
               float)j) - 1] *
            x2i->data[(int)(1.0F + (float)j) - 1]) / (
               float)fcnOutput;

/* sum(x(1,:).*conj(x(2,:))); % Computing the real part
   of C12 from the Received Symbols */
*c12i += (x1i->data[(int)(1.0F + (float)j) - 1] * x2r->
   data[(int)(1.0F +
            (float)j) - 1] - x1r->data[(int)(1.0F + (
               float)j) - 1] *
            x2i->data[(int)(1.0F + (float)j) - 1]) / (
               float)fcnOutput;

/* sum(x(1,:).*conj(x(2,:))); % Computing the imaginary
   part of C12 from the Received Symbols */
*c22 += (x2r->data[(int)(1.0F + (float)j) - 1] * x2r->
   data[(int)(1.0F +
            (float)j) - 1] + x2i->data[(int)(1.0F + (float
               )j) - 1] * x2i->
```

```
                data[(int)(1.0F + (float)j) - 1]) / (float)
                    fcnOutput;

    /*  Computing C22 from the Received Symbols */
    /* c=(2/n)*[c11 c12;c21 c22]; % Covariance matrix of the
        Received Symbols */
  }
}
```

B.2 Channel Coefficient Computation

```
/* Include Files */
#include "rt_nonfinite.h"
#include "c_comp.h"
#include "h_comp.h"
#include "r_gen.h"
#include <stdio.h>

/* Function Definitions */

/*
 * Arguments    : float c11
 *                float c12r
 *                float c12i
 *                float c22
 *                float *h1
 *                float *h2
 *                float *s_angh
```

```
 * Return Type  : void
 */
void h_comp(float c11, float c12r, float c12i, float c22,
   float *h1, float *h2,
            float *s_angh)
{
  float a;
  float a2;
  float z;
  float p;

  /*  Ratio of the Powers radiated by the two antennas */
  *h1 = (float)sqrt((c11 - 0.6F * c22) / 0.64F);

  /*  Computing the Magnitude of Channel Coefficient h1 */
  *h2 = (float)sqrt((c22 - 0.6F * c11) / 0.64F);

  /*  Computing the Magnitude of Channel Coefficient h2 */
  /* s_angh=atan2_fixpt(c12i,c12r); % Computing the Sum of
     the phase shifts of the channel coefficients */
  /* % Computing the sum of the angles of the channel
     characteristics */
  *s_angh = 0.0F;
  a = c12i / c12r;
  if (a < 0.0F) {
    if (-a > 1.0F) {
      a2 = 1.0F / -a;
      z = a2 / (1.0F + 0.28125F * a2 * a2);
```

```
    if ((c12r > 0.0F) && (c12i < 0.0F)) {
      *s_angh = -(1.57079637F - z);
    } else {
      if ((c12r < 0.0F) && (c12i > 0.0F)) {
        *s_angh = 3.14159274F - (1.57079637F - z);
      }
    }
  } else if (-a < 1.0F) {
    p = -a / (1.0F + 0.28125F * -a * -a);
    if ((c12r > 0.0F) && (c12i < 0.0F)) {
      *s_angh = -p;
    } else {
      if ((c12r < 0.0F) && (c12i > 0.0F)) {
        *s_angh = 3.14159274F - p;
      }
    }
  } else if ((c12r > 0.0F) && (c12i < 0.0F)) {
    *s_angh = -0.785398185F;
  } else {
    if ((c12r < 0.0F) && (c12i > 0.0F)) {
      *s_angh = 2.3561945F;
    }
  }
} else if (a > 0.0F) {
  if (a > 1.0F) {
    a2 = 1.0F / a;
    z = a2 / (1.0F + 0.28125F * a2 * a2);
    if ((c12r > 0.0F) && (c12i > 0.0F)) {
```

```
      *s_angh = 1.57079637F - z;
    } else {
      if ((c12r < 0.0F) && (c12i < 0.0F)) {
        *s_angh = -3.14159274F + (1.57079637F - z);
      }
    }
  } else if (a < 1.0F) {
    p = a / (1.0F + 0.28125F * a * a);
    if ((c12r > 0.0F) && (c12i > 0.0F)) {
      *s_angh = p;
    } else {
      if ((c12r < 0.0F) && (c12i < 0.0F)) {
        *s_angh = -3.14159274F + p;
      }
    }
  } else if ((c12r > 0.0F) && (c12i > 0.0F)) {
    *s_angh = 0.785398185F;C Code for SOS-based Blind
       Estimation
  } else {
    if ((c12r < 0.0F) && (c12i < 0.0F)) {
      *s_angh = -2.3561945F;
    }
  }
} else if (a == 0.0F) {
  if (((c12r > 0.0F) && (c12i == 0.0F)) || (!(c12r < 0.0F)
     ) || (!(c12i == 0.0F)))
  {
  } else {
```

```
      *s_angh = 3.14159274F;
    }
  } else if ((c12i > 0.0F) && (c12r == 0.0F)) {
    *s_angh = 1.57079637F;
  } else {
    if ((c12i < 0.0F) && (c12r == 0.0F)) {
      *s_angh = -1.57079637F;
    }
  }
printf("|h1|= %f;  |h2|= %f ; Sum of Phases = %f", *h1, *h2,
    *s_angh );

}
```

Appendix C

VHDL Code for SOS-based Blind Estimation

C.1 Expected Value of Covariance Matrix Computation

```
--
-- -------------------------------------------------------------

-- Rate and Clocking Details
--
-- -------------------------------------------------------------

-- Model base rate: 1e-06
-- Target subsystem base rate: 1e-06
--
--
```

```
-- -------------------------------------------------------------
--
-- -------------------------------------------------------------
--
-- Module: covar_comp
-- Source Path: block_diagram/Subsystem
-- Hierarchy Level: 0
--
--
-- -------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY Subsystem IS
  PORT( x1r        :   IN    std_logic_vector(7 DOWNTO 0);
    -- int8
        x1r1       :   IN    std_logic_vector(7 DOWNTO 0);
          -- int8
        x1r2       :   IN    std_logic_vector(7 DOWNTO 0);
          -- int8
        x2r        :   IN    std_logic_vector(7 DOWNTO 0);
          -- int8
```

```
        C11        :   OUT   std_logic_vector(31 DOWNTO 0);
          -- int32
        C12r       :   OUT   std_logic_vector(31 DOWNTO 0);
          -- int32
        C12i       :   OUT   std_logic_vector(31 DOWNTO 0);
          -- int32
        C22        :   OUT   std_logic_vector(31 DOWNTO 0)
          -- int32
        );
END Subsystem;


ARCHITECTURE rtl OF Subsystem IS

  -- Signals
  SIGNAL x1r_signed                       : signed(7 DOWNTO
     0); -- int8
  SIGNAL Product_out1                     : signed(15 DOWNTO
      0); -- int16
  SIGNAL x1r1_signed                      : signed(7 DOWNTO
     0); -- int8
  SIGNAL Product1_out1                    : signed(15 DOWNTO
      0); -- int16
  SIGNAL Sum_out1                         : signed(16 DOWNTO
      0); -- sfix17
  SIGNAL Sum_out1_dtc                     : signed(15 DOWNTO
      0); -- sfix16
  SIGNAL Sum_of_Elements_out1             : signed(15 DOWNTO
```

```
    0);  -- int16
SIGNAL alpha_out1                 : signed(15 DOWNTO
    0);  -- int16
SIGNAL Product8_out1              : signed(31 DOWNTO
    0);  -- int32
SIGNAL x1r2_signed                : signed(7 DOWNTO
   0);  -- int8
SIGNAL Product4_out1              : signed(15 DOWNTO
    0);  -- int16
SIGNAL x2r_signed                 : signed(7 DOWNTO
   0);  -- int8
SIGNAL Product5_out1              : signed(15 DOWNTO
    0);  -- int16
SIGNAL Sum4_out1                  : signed(16 DOWNTO
    0);  -- sfix17
SIGNAL Sum4_out1_dtc              : signed(15 DOWNTO
    0);  -- sfix16
SIGNAL Sum_of_Elements1_out1      : signed(15 DOWNTO
    0);  -- int16
SIGNAL alpha_1_out1               : signed(15 DOWNTO
    0);  -- int16
SIGNAL Product9_out1              : signed(31 DOWNTO
    0);  -- int32
SIGNAL Product7_out1              : signed(15 DOWNTO
    0);  -- int16
SIGNAL Product6_out1              : signed(15 DOWNTO
    0);  -- int16
SIGNAL Sum6_out1                  : signed(16 DOWNTO
```

```
    0);  -- sfix17
  SIGNAL Sum6_out1_dtc                    : signed(15 DOWNTO
    0);  -- sfix16
  SIGNAL Sum_of_Elements2_out1            : signed(15 DOWNTO
    0);  -- int16
  SIGNAL alpha_2_out1                     : signed(15 DOWNTO
    0);  -- int16
  SIGNAL Product10_out1                   : signed(31 DOWNTO
    0);  -- int32
  SIGNAL Product2_out1                    : signed(15 DOWNTO
    0);  -- int16
  SIGNAL Product3_out1                    : signed(15 DOWNTO
    0);  -- int16
  SIGNAL Sum2_out1                        : signed(16 DOWNTO
    0);  -- sfix17
  SIGNAL Sum2_out1_dtc                    : signed(15 DOWNTO
    0);  -- sfix16
  SIGNAL Sum_of_Elements3_out1            : signed(15 DOWNTO
    0);  -- int16
  SIGNAL alpha_3_out1                     : signed(15 DOWNTO
    0);  -- int16
  SIGNAL Product11_out1                   : signed(31 DOWNTO
    0);  -- int32

BEGIN
  x1r_signed <= signed(x1r);

  Product_out1 <= x1r_signed * x1r_signed;
```

```
x1r1_signed <= signed(x1r1);

Product1_out1 <= x1r1_signed * x1r1_signed;

Sum_out1 <= resize(Product_out1, 17) + resize(
    Product1_out1, 17);

Sum_out1_dtc <= Sum_out1(15 DOWNTO 0);

Sum_of_Elements_out1 <= Sum_out1_dtc;

alpha_out1 <= to_signed(16#0000#, 16);

Product8_out1 <= Sum_of_Elements_out1 * alpha_out1;

C11 <= std_logic_vector(Product8_out1);

x1r2_signed <= signed(x1r2);

Product4_out1 <= x1r_signed * x1r2_signed;

x2r_signed <= signed(x2r);

Product5_out1 <= x1r1_signed * x2r_signed;

Sum4_out1 <= resize(Product4_out1, 17) + resize(
    Product5_out1, 17);
```

```
Sum4_out1_dtc <= Sum4_out1(15 DOWNTO 0);

Sum_of_Elements1_out1 <= Sum4_out1_dtc;

alpha_1_out1 <= to_signed(16#0000#, 16);

Product9_out1 <= Sum_of_Elements1_out1 * alpha_1_out1;

C12r <= std_logic_vector(Product9_out1);

Product7_out1 <= x1r1_signed * x1r2_signed;

Product6_out1 <= x1r_signed * x2r_signed;

Sum6_out1 <= resize(Product7_out1, 17) - resize(
   Product6_out1, 17);

Sum6_out1_dtc <= Sum6_out1(15 DOWNTO 0);

Sum_of_Elements2_out1 <= Sum6_out1_dtc;

alpha_2_out1 <= to_signed(16#0000#, 16);

Product10_out1 <= Sum_of_Elements2_out1 * alpha_2_out1;

C12i <= std_logic_vector(Product10_out1);
```

```
  Product2_out1 <= x1r2_signed * x1r2_signed;

  Product3_out1 <= x2r_signed * x2r_signed;

  Sum2_out1 <= resize(Product2_out1, 17) + resize(
     Product3_out1, 17);

  Sum2_out1_dtc <= Sum2_out1(15 DOWNTO 0);

  Sum_of_Elements3_out1 <= Sum2_out1_dtc;

  alpha_3_out1 <= to_signed(16#0000#, 16);

  Product11_out1 <= Sum_of_Elements3_out1 * alpha_3_out1;

  C22 <= std_logic_vector(Product11_out1);

END rtl;
```

C.2 Channel Coefficient Computation

```
--
   ----------------------------------------------------------------


-- Rate and Clocking Details
--
   ----------------------------------------------------------------
```

```
-- Design base rate: 1
--
--
    ------------------------------------------------------------------------




--
    ------------------------------------------------------------------------


--
-- Module: h_comp_fixpt
-- Source Path: h_comp_fixpt
-- Hierarchy Level: 0
--
--
    ------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
USE work.h_comp_fixpt_pkg.ALL;

ENTITY h_comp_fixpt IS
  PORT( c11                               :   IN    std_logic_vector(13
    DOWNTO 0);  -- ufix14_En13
        c12r                              :   IN    std_logic_vector(13
          DOWNTO 0);  -- sfix14_En17
```

```
        c12i                    :   IN    std_logic_vector(13
          DOWNTO 0);  -- ufix14_En16
        c22                     :   IN    std_logic_vector(13
          DOWNTO 0);  -- ufix14_En14
        ah1                     :   OUT   std_logic_vector(13
          DOWNTO 0);  -- ufix14_En14
        ah2                     :   OUT   std_logic_vector(13
          DOWNTO 0);  -- ufix14_En14
        as_angh                 :   OUT   std_logic_vector(13
          DOWNTO 0)  -- ufix14_En13
        );
END h_comp_fixpt;


ARCHITECTURE rtl OF h_comp_fixpt IS

  -- Constants
  CONSTANT C_divbyzero_p                  : unsigned(91
    DOWNTO 0) :=
    unsigned'(X"FFFFFFFFFFFFFFFFFFFFFFF");  -- ufix92
  CONSTANT One                            : unsigned(35
    DOWNTO 0) :=
    unsigned'(X"800000000");  -- ufix36
  CONSTANT c_divbyzero_p_0                : unsigned(35
    DOWNTO 0) :=
    unsigned'(X"FFFFFFFFF");  -- ufix36
  CONSTANT c_divbyzero_p_1_1              : signed(39 DOWNTO
      0) :=
```

```
    signed'(X"7FFFFFFFFF");  -- sfix40
  CONSTANT C_divbyzero_n                    : signed(39 DOWNTO
      0) :=
    signed'(X"8000000000");  -- sfix40

  -- Signals
  SIGNAL c11_unsigned        : unsigned(13 DOWNTO 0);  --
      ufix14_En13
  SIGNAL c12r_signed         : signed(13 DOWNTO 0);  --
      sfix14_En17
  SIGNAL c12i_unsigned       : unsigned(13 DOWNTO 0);  --
      ufix14_En16
  SIGNAL c22_unsigned        : unsigned(13 DOWNTO 0);  --
      ufix14_En14
  SIGNAL ah1_tmp             : unsigned(13 DOWNTO 0);  --
      ufix14_En14
  SIGNAL ah2_tmp             : unsigned(13 DOWNTO 0);  --
      ufix14_En14
  SIGNAL as_angh_tmp         : unsigned(13 DOWNTO 0);  --
      ufix14_En13

BEGIN
  c11_unsigned <= unsigned(c11);

  c12r_signed <= signed(c12r);

  c12i_unsigned <= unsigned(c12i);
```

```
c22_unsigned <= unsigned(c22);

h_comp_fixpt_1_output : PROCESS (c11_unsigned, c12r_signed
    , c12i_unsigned,
c22_unsigned)
  VARIABLE c1 : signed(60 DOWNTO 0);
  VARIABLE c1_0 : signed(59 DOWNTO 0);
  VARIABLE tmp : unsigned(13 DOWNTO 0);
  VARIABLE a : signed(13 DOWNTO 0);
  VARIABLE a1 : unsigned(13 DOWNTO 0);
  VARIABLE a2 : unsigned(13 DOWNTO 0);
  VARIABLE a1_0 : unsigned(13 DOWNTO 0);
  VARIABLE a2_0 : unsigned(13 DOWNTO 0);
  VARIABLE tmp_0 : std_logic;
  VARIABLE ytemp : signed(30 DOWNTO 0);
  VARIABLE y : signed(30 DOWNTO 0);
  VARIABLE ytemp_0 : signed(29 DOWNTO 0);
  VARIABLE y_0 : signed(29 DOWNTO 0);
  VARIABLE p : unsigned(13 DOWNTO 0);
  VARIABLE a2_1 : unsigned(13 DOWNTO 0);
  VARIABLE p_0 : unsigned(13 DOWNTO 0);
  VARIABLE tmp_1 : unsigned(13 DOWNTO 0);
  VARIABLE p_1 : unsigned(13 DOWNTO 0);
  VARIABLE a2_2 : unsigned(13 DOWNTO 0);
  VARIABLE p_2 : unsigned(13 DOWNTO 0);
  VARIABLE tmp_2 : unsigned(13 DOWNTO 0);
  VARIABLE div_temp : signed(58 DOWNTO 0);
  VARIABLE div_temp_0 : signed(58 DOWNTO 0);
```

```
VARIABLE div_temp_1 : signed(39 DOWNTO 0);
VARIABLE div_temp_2 : unsigned(35 DOWNTO 0);
VARIABLE add_temp : unsigned(48 DOWNTO 0);
VARIABLE div_temp_3 : unsigned(91 DOWNTO 0);
VARIABLE add_temp_0 : unsigned(48 DOWNTO 0);
VARIABLE div_temp_4 : unsigned(91 DOWNTO 0);
VARIABLE div_temp_5 : unsigned(35 DOWNTO 0);
VARIABLE add_temp_1 : unsigned(48 DOWNTO 0);
VARIABLE div_temp_6 : unsigned(91 DOWNTO 0);
VARIABLE add_temp_2 : unsigned(48 DOWNTO 0);
VARIABLE div_temp_7 : unsigned(91 DOWNTO 0);
VARIABLE sub_cast : unsigned(29 DOWNTO 0);
VARIABLE mul_temp : unsigned(27 DOWNTO 0);
VARIABLE sub_cast_0 : unsigned(29 DOWNTO 0);
VARIABLE sub_temp : unsigned(29 DOWNTO 0);
VARIABLE cast : signed(30 DOWNTO 0);
VARIABLE cast_0 : signed(58 DOWNTO 0);
VARIABLE cast_1 : vector_of_unsigned8(0 TO 29);
VARIABLE sll_temp : vector_of_signed31(0 TO 29);
VARIABLE sub_cast_1 : unsigned(28 DOWNTO 0);
VARIABLE mul_temp_0 : unsigned(27 DOWNTO 0);
VARIABLE sub_cast_2 : unsigned(28 DOWNTO 0);
VARIABLE sub_temp_0 : unsigned(28 DOWNTO 0);
VARIABLE cast_2 : signed(29 DOWNTO 0);
VARIABLE cast_3 : signed(58 DOWNTO 0);
VARIABLE mul_temp_1 : vector_of_signed62(0 TO 29);
VARIABLE cast_4 : vector_of_signed62(0 TO 29);
VARIABLE cast_5 : vector_of_unsigned8(0 TO 28);
```

```
VARIABLE sll_temp_0 : vector_of_signed30(0 TO 28);
VARIABLE mul_temp_2 : vector_of_signed60(0 TO 28);
VARIABLE cast_6 : signed(18 DOWNTO 0);
VARIABLE cast_7 : unsigned(16 DOWNTO 0);
VARIABLE cast_8 : signed(18 DOWNTO 0);
VARIABLE cast_9 : unsigned(16 DOWNTO 0);
VARIABLE cast_10 : signed(18 DOWNTO 0);
VARIABLE cast_11 : unsigned(16 DOWNTO 0);
VARIABLE cast_12 : signed(18 DOWNTO 0);
VARIABLE cast_13 : unsigned(16 DOWNTO 0);
VARIABLE cast_14 : unsigned(16 DOWNTO 0);
VARIABLE cast_15 : signed(18 DOWNTO 0);
VARIABLE slice_cast : signed(14 DOWNTO 0);
VARIABLE slice_cast_0 : signed(39 DOWNTO 0);
VARIABLE slice_cast_1 : signed(39 DOWNTO 0);
VARIABLE cast_16 : signed(14 DOWNTO 0);
VARIABLE cast_17 : signed(39 DOWNTO 0);
VARIABLE cast_18 : signed(37 DOWNTO 0);
VARIABLE cast_19 : signed(14 DOWNTO 0);
VARIABLE cast_20 : signed(15 DOWNTO 0);
VARIABLE cast_21 : signed(15 DOWNTO 0);
VARIABLE cast_22 : signed(14 DOWNTO 0);
VARIABLE mul_temp_3 : unsigned(27 DOWNTO 0);
VARIABLE mul_temp_4 : unsigned(41 DOWNTO 0);
VARIABLE add_cast : unsigned(48 DOWNTO 0);
VARIABLE cast_23 : unsigned(27 DOWNTO 0);
VARIABLE cast_24 : unsigned(91 DOWNTO 0);
VARIABLE mul_temp_5 : unsigned(27 DOWNTO 0);
```

```
VARIABLE mul_temp_6 : unsigned(41 DOWNTO 0);
VARIABLE add_cast_0 : unsigned(48 DOWNTO 0);
VARIABLE cast_25 : unsigned(62 DOWNTO 0);
VARIABLE cast_26 : unsigned(91 DOWNTO 0);
VARIABLE cast_27 : signed(18 DOWNTO 0);
VARIABLE cast_28 : unsigned(16 DOWNTO 0);
VARIABLE sub_cast_3 : unsigned(62 DOWNTO 0);
VARIABLE sub_cast_4 : unsigned(63 DOWNTO 0);
VARIABLE sub_temp_1 : unsigned(63 DOWNTO 0);
VARIABLE mul_temp_7 : unsigned(27 DOWNTO 0);
VARIABLE mul_temp_8 : unsigned(41 DOWNTO 0);
VARIABLE add_cast_1 : unsigned(48 DOWNTO 0);
VARIABLE cast_29 : unsigned(27 DOWNTO 0);
VARIABLE t_0 : signed(16 DOWNTO 0);
VARIABLE sub_cast_5 : unsigned(13 DOWNTO 0);
VARIABLE cast_30 : signed(18 DOWNTO 0);
VARIABLE cast_31 : unsigned(16 DOWNTO 0);
VARIABLE cast_32 : unsigned(91 DOWNTO 0);
VARIABLE mul_temp_9 : unsigned(27 DOWNTO 0);
VARIABLE mul_temp_10 : unsigned(41 DOWNTO 0);
VARIABLE add_cast_2 : unsigned(48 DOWNTO 0);
VARIABLE t_1 : unsigned(15 DOWNTO 0);
VARIABLE sub_cast_6 : unsigned(13 DOWNTO 0);
VARIABLE cast_33 : unsigned(62 DOWNTO 0);
VARIABLE cast_34 : unsigned(91 DOWNTO 0);
VARIABLE sub_cast_7 : unsigned(62 DOWNTO 0);
VARIABLE sub_cast_8 : signed(63 DOWNTO 0);
VARIABLE sub_temp_2 : signed(63 DOWNTO 0);
```

```
  VARIABLE cast_35 : signed(18 DOWNTO 0);
  VARIABLE cast_36 : unsigned(16 DOWNTO 0);
  VARIABLE cast_37 : signed(18 DOWNTO 0);
  VARIABLE cast_38 : unsigned(16 DOWNTO 0);
BEGIN

  sub_cast := resize(c11_unsigned & '0' & '0' & '0' & '0'
    & '0' & '0'
   & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0',
      30);
  mul_temp := to_unsigned(16#2666#, 14) * c22_unsigned;
  sub_cast_0 := resize(mul_temp, 30);
  sub_temp := sub_cast - sub_cast_0;
  cast := signed(resize(sub_temp, 31));
  cast_0 := cast & '0' & '0' & '0' & '0' & '0' & '0' & '0'
   & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0'
   & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
       '0'
    & '0' & '0';
  div_temp := cast_0 / to_signed(16#0A3D8F5C#, 31);
  c1 := resize(div_temp, 61);
  IF c1 <= to_signed(0, 61) THEN
    y := to_signed(16#00000000#, 31);
  ELSE
    y := to_signed(16#00000000#, 31);

    FOR i IN 29 DOWNTO 0 LOOP
      cast_1(i) := unsigned(to_signed(i, 32)(7 DOWNTO 0));
```

```
      sll_temp(i) := to_signed(16#00000001#, 31) sll
         to_integer
      (cast_1(i));
      ytemp := y OR sll_temp(i);
      mul_temp_1(i) := ytemp * ytemp;
      cast_4(i) := resize(c1, 62);
      IF mul_temp_1(i) <= cast_4(i) THEN
        y := ytemp;
      END IF;
    END LOOP;

END IF;
-- Computing the Magnitude of Channel Coefficient h1
sub_cast_1 := resize(c22_unsigned & '0' & '0' & '0' &
    '0' & '0'
 & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0', 29);
mul_temp_0 := to_unsigned(16#2666#, 14) * c11_unsigned;
sub_cast_2 := resize(mul_temp_0, 29);
sub_temp_0 := sub_cast_1 - sub_cast_2;
cast_2 := signed(resize(sub_temp_0, 30));
cast_3 := cast_2 & '0' & '0' & '0' & '0' & '0' & '0' &
    '0' & '0'
 & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
      '0' &
  '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
     '0' & '0';
div_temp_0 := cast_3 / to_signed(16#0A3D8F5C#, 31);
c1_0 := resize(div_temp_0, 60);
```

```
IF c1_0 <= to_signed(0, 60) THEN
  y_0 := to_signed(16#00000000#, 30);
ELSE
  y_0 := to_signed(16#00000000#, 30);

  FOR i_0 IN 28 DOWNTO 0 LOOP
    cast_5(i_0) := unsigned(to_signed(i_0, 32)(7 DOWNTO
      0));
    sll_temp_0(i_0) := to_signed(16#00000001#, 30) sll
      to_integer
    (cast_5(i_0));
    ytemp_0 := y_0 OR sll_temp_0(i_0);
    mul_temp_2(i_0) := ytemp_0 * ytemp_0;
    IF mul_temp_2(i_0) <= c1_0 THEN
      y_0 := ytemp_0;
    END IF;
  END LOOP;

END IF;
-- Computing the Magnitude of Channel Coefficient h2
--s_angh=atan2_fixpt(c12i,c12r); % Computing the Sum of
    the phase shifts of the channel coefficients
--% Computing the sum of the angles of the channel
    characteristics
cast_6 := resize(c12r_signed, 19);
cast_7 := resize(c12i_unsigned, 17);
IF (cast_6 < to_signed(16#00000#, 19)) AND (cast_7 >
   to_unsigned(16#00000#, 17)) THEN
```

```
    tmp_1 := to_unsigned(16#0B65#, 14);
ELSE
    tmp_1 := to_unsigned(16#0000#, 14);
END IF;
tmp_2 := to_unsigned(16#0000#, 14);
cast_8 := resize(c12r_signed, 19);
cast_9 := resize(c12i_unsigned, 17);
IF (cast_8 > to_signed(16#00000#, 19)) AND (cast_9 >
    to_unsigned(16#00000#, 17)) THEN
    tmp_2 := to_unsigned(16#1921#, 14);
END IF;
cast_10 := resize(c12r_signed, 19);
cast_11 := resize(c12i_unsigned, 17);
IF (cast_10 < to_signed(16#00000#, 19)) AND (cast_11 =
    to_unsigned(16#00000#, 17)) THEN
    tmp := to_unsigned(16#2487#, 14);
ELSE
    tmp := to_unsigned(16#0000#, 14);
END IF;
cast_12 := resize(c12r_signed, 19);
cast_13 := resize(c12i_unsigned, 17);
IF (cast_12 > to_signed(16#00000#, 19)) AND (cast_13 =
    to_unsigned(16#00000#, 17)) THEN
    tmp := to_unsigned(16#0000#, 14);
END IF;
as_angh_tmp <= to_unsigned(16#0000#, 14);
cast_14 := resize(c12i_unsigned, 17);
cast_15 := resize(c12r_signed, 19);
```

```
IF (cast_14 > to_unsigned(16#00000#, 17)) AND (cast_15 =
    to_signed(16#00000#, 19)) THEN
  as_angh_tmp <= to_unsigned(16#3243#, 14);
END IF;
slice_cast := signed(resize(c12i_unsigned, 15));
slice_cast_0 := slice_cast & '0' & '0' & '0' & '0' & '0'
    & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0'
    & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0'
    & '0' & '0';
IF slice_cast_0(39) = c12r_signed(13) THEN
  slice_cast_1 := c_divbyzero_p_1_1;
ELSE
  slice_cast_1 := C_divbyzero_n;
END IF;
cast_16 := signed(resize(c12i_unsigned, 15));
cast_17 := cast_16 & '0' & '0' & '0' & '0' & '0' & '0' &
    '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
    '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
    '0';
IF c12r_signed = 0 THEN
  div_temp_1 := slice_cast_1;
ELSE
  div_temp_1 := cast_17 / c12r_signed;
END IF;
cast_18 := div_temp_1(37 DOWNTO 0);
a := cast_18(27 DOWNTO 14);
IF a = to_signed(16#0000#, 14) THEN
  as_angh_tmp <= tmp;
```

```
END IF;
cast_19 := resize(a, 15);
cast_20 := resize(cast_19, 16);
cast_21 := -(cast_20);
cast_22 := cast_21(14 DOWNTO 0);
a1 := unsigned(cast_22(12 DOWNTO 0) & '0');
IF a1 = to_unsigned(16#0800#, 14) THEN
  NULL;
ELSE
  tmp_1 := to_unsigned(16#0000#, 14);
END IF;
a2 := a1(8 DOWNTO 0) & '0' & '0' & '0' & '0' & '0';
a1_0 := unsigned(a(12 DOWNTO 0) & '0');
IF a1_0 = to_unsigned(16#0800#, 14) THEN
  NULL;
ELSE
  tmp_2 := to_unsigned(16#0000#, 14);
END IF;
a2_0 := a1_0(8 DOWNTO 0) & '0' & '0' & '0' & '0' & '0';
IF a < to_signed(16#0000#, 14) THEN
  tmp_0 := '1';
ELSE
  tmp_0 := '0';
END IF;
IF tmp_0 = '1' THEN
  IF a1 > to_unsigned(16#0800#, 14) THEN
    IF a1 = 0 THEN
      div_temp_2 := c_divbyzero_p_0;
```

```
ELSE
  div_temp_2 := One / a1;
END IF;
cast_23 := div_temp_2(27 DOWNTO 0);
a2_1 := cast_23(21 DOWNTO 8);
mul_temp_5 := to_unsigned(16#2400#, 14) * a2_1;
mul_temp_6 := mul_temp_5 * a2_1;
add_cast_0 := resize(mul_temp_6, 49);
add_temp_0 := unsigned'("
   01000000000000000000000000000000000000000000000000"
   ) + add_cast_0;
cast_26 := a2_1 & '0' & '0' & '0' & '0' & '0' & '0'
   & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
    '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
    '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
    '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
    '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
    '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
    '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
    '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
    '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0';
IF add_temp_0 = 0 THEN
  div_temp_4 := C_divbyzero_p;
ELSE
  div_temp_4 := cast_26 / add_temp_0;
END IF;
sub_cast_3 := div_temp_4(62 DOWNTO 0);
sub_cast_4 := resize(sub_cast_3, 64);
```

```
    sub_temp_1 := unsigned'(X"0000C90C00000000") -
        sub_cast_4;
    p_0 := sub_temp_1(47 DOWNTO 34);
    cast_30 := resize(c12r_signed, 19);
    cast_31 := resize(c12i_unsigned, 17);
    IF (cast_30 < to_signed(16#00000#, 19)) AND (cast_31
        > to_unsigned(16#00000#, 17)) THEN
      t_1 := to_unsigned(16#6486#, 16);
      sub_cast_6 := t_1(13 DOWNTO 0);
      tmp_1 := sub_cast_6 - p_0;
    ELSE
      tmp_1 := to_unsigned(16#0000#, 14);
    END IF;
  ELSIF a1 < to_unsigned(16#0800#, 14) THEN
    mul_temp_3 := to_unsigned(16#2400#, 14) * a2;
    mul_temp_4 := mul_temp_3 * a2;
    add_cast := resize(mul_temp_4, 49);
    add_temp := unsigned'("
       01000000000000000000000000000000000000000000000000"
       ) + add_cast;
    cast_24 := a2 & '0' & '0' & '0' & '0' & '0' & '0' &
       '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
       '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
       '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
       '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
       '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
       '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
       '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
```

```
        '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
        '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0';
    IF add_temp = 0 THEN
      div_temp_3 := C_divbyzero_p;
    ELSE
      div_temp_3 := cast_24 / add_temp;
    END IF;
    cast_25 := div_temp_3(62 DOWNTO 0);
    p := cast_25(47 DOWNTO 34);
    cast_27 := resize(c12r_signed, 19);
    cast_28 := resize(c12i_unsigned, 17);
    IF (cast_27 < to_signed(16#00000#, 19)) AND (cast_28
        > to_unsigned(16#00000#, 17)) THEN
      t_0 := to_signed(16#06486#, 17);
      sub_cast_5 := unsigned(t_0(13 DOWNTO 0));
      tmp_1 := sub_cast_5 - p;
    ELSE
      tmp_1 := to_unsigned(16#0000#, 14);
    END IF;
  END IF;
ELSE
  tmp_1 := to_unsigned(16#0000#, 14);
END IF;
IF tmp_0 = '1' THEN
  as_angh_tmp <= to_unsigned(16#0000#, 14);
ELSIF a > to_signed(16#0000#, 14) THEN
  IF a1_0 > to_unsigned(16#0800#, 14) THEN
    IF a1_0 = 0 THEN
```

```
    div_temp_5 := c_divbyzero_p_0;
ELSE
    div_temp_5 := One / a1_0;
END IF;
cast_29 := div_temp_5(27 DOWNTO 0);
a2_2 := cast_29(21 DOWNTO 8);
mul_temp_9 := to_unsigned(16#2400#, 14) * a2_2;
mul_temp_10 := mul_temp_9 * a2_2;
add_cast_2 := resize(mul_temp_10, 49);
add_temp_2 := unsigned'("
    01000000000000000000000000000000000000000000000000"
    ) + add_cast_2;
cast_34 := a2_2 & '0' & '0' & '0' & '0' & '0' & '0'
    & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
    '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
    '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
    '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
    '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
    '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
    '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
    '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
    '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0';
IF add_temp_2 = 0 THEN
    div_temp_7 := C_divbyzero_p;
ELSE
    div_temp_7 := cast_34 / add_temp_2;
END IF;
sub_cast_7 := div_temp_7(62 DOWNTO 0);
```

```
    sub_cast_8 := signed(resize(sub_cast_7, 64));
    sub_temp_2 := signed'(X"0000C90C00000000") -
        sub_cast_8;
    p_2 := unsigned(sub_temp_2(47 DOWNTO 34));
    tmp_2 := to_unsigned(16#0000#, 14);
    cast_37 := resize(c12r_signed, 19);
    cast_38 := resize(c12i_unsigned, 17);
    IF (cast_37 > to_signed(16#00000#, 19)) AND (cast_38
        > to_unsigned(16#00000#, 17)) THEN
      tmp_2 := p_2;
    END IF;
  ELSIF a1_0 < to_unsigned(16#0800#, 14) THEN
    mul_temp_7 := to_unsigned(16#2400#, 14) * a2_0;
    mul_temp_8 := mul_temp_7 * a2_0;
    add_cast_1 := resize(mul_temp_8, 49);
    add_temp_1 := unsigned'("
        01000000000000000000000000000000000000000000000000"
        ) + add_cast_1;
    cast_32 := a2_0 & '0' & '0' & '0' & '0' & '0' & '0'
       & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
        '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
        '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
        '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
        '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
        '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
        '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
        '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
        '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0';
```

```
        IF add_temp_1 = 0 THEN
          div_temp_6 := C_divbyzero_p;
        ELSE
          div_temp_6 := cast_32 / add_temp_1;
        END IF;
        cast_33 := div_temp_6(62 DOWNTO 0);
        p_1 := cast_33(47 DOWNTO 34);
        tmp_2 := to_unsigned(16#0000#, 14);
        cast_35 := resize(c12r_signed, 19);
        cast_36 := resize(c12i_unsigned, 17);
        IF (cast_35 > to_signed(16#00000#, 19)) AND (cast_36
            > to_unsigned(16#00000#, 17)) THEN
          tmp_2 := p_1;
        END IF;
      END IF;
      as_angh_tmp <= tmp_2;
    END IF;
    IF tmp_0 = '1' THEN
      as_angh_tmp <= tmp_1;
    END IF;
    ah1_tmp <= unsigned(y(13 DOWNTO 0));
    ah2_tmp <= unsigned(y_0(13 DOWNTO 0));
END PROCESS h_comp_fixpt_1_output;


ah1 <= std_logic_vector(ah1_tmp);

ah2 <= std_logic_vector(ah2_tmp);
```

```
    as_angh <= std_logic_vector(as_angh_tmp);

END rtl;
```

Bibliography

[1] S. M. Alamouti, "A Simple Transmit Diversity Technique for Wireless Communications", *IEEE Journal on Selected Areas in Communications.* , 16(8), 1451-1458. Oct 1998

[2] A. Goldsmith, *Wireless communications.* Cambridge: Cambridge University Press, 2005.

[3] A. Belouchrani, K. Abed-Meraim, J. Cardoso., E. Moulines, "A blind source separation technique using second-order statistics". *IEEE Transactions on Signal Processing*, 45(2), 434-444, Feb 1997

[4] D. G. Brennan," Linear diversity combining techniques", Proceedings of the IEEE *Proceedings of the IEEE,*91(2), 331-356, 2003

[5] P. M. Castro,A. Dapena, J. A. García-Naya, J. Labrador, "A low-cost decision-aided channel estimation method for alamouti OSTBC", *Neural Computing and Applications*, 23(6), 1597-1604, Nov 2013

[6] Y. A. Eldemerdash, O. A. Dobre, B. J. Liao, "Blind identification of SM and alamouti STBC-OFDM signals", *IEEE Transactions on Wireless Communications*, 14(2), 972-982, Feb 2015

[7] T. P. Krauss, R. D. Zoltowski, "Bilinear approach to multiuser second-order statistics-based blind channel estimation", *IEEE Transactions on Signal Processing*, 48(9), 2473-2486, Sep 2000

[8] R. Lyons, "Another contender in the arctangent race", *IEEE Signal Processing Magazine*, 21(1), 109-110, 2012

[9] L. Parra, P. Sajda, "Blind source separation via generalized eigenvalue decomposition", *Journal of Machine Learning Research*, 4(7-8), 1261-1269, Oct 2004.

[10] H. J. Pérez-Iglesias, J. A. Garcia-Naya, A. Dapena," A blind channel estimation strategy for the 2X1 alamouti system based on diagonalising 4th-order cumulant matrices", *IEEE International Conference on Acoustics, Speech and Signal Processing* , 3329-3332, 2008

[11] H. J. Pérez-Iglesias, A. Dapena, L. Castedo, V. Zarzoso, "Blind channel identification for Alamouti's coding systems based on eigenvector decomposition", *Proceedings of European Wireless*, April 2007

[12] J. E. Volder, "The CORDIC trigonometric computing technique", *IRE Transactions on Electronic Computers*, EC-8(3), 330-334, 1959

[13] L. Zhou, J. K. Zhang, K. M. Wong . "Blind unique identification of Alamouti space-time coded channel via signal design and transmission technique", *ISSPA*, 691-694, August 2005