.

Wireless Extension to the Existing SystemC Design Methodology

by
Ibrahim Asaad Aref

A Thesis presented for the degree of

Doctor of Philosophy

to the Department of Electronics and Electrical Engineering,

College of Science and Engineering, School of Engineering,
University of Glasgow

UNIVERSITY
of
GLASGOW

February 2011

.

.

Wireless Extension to the Existing SystemC Design Methodology

Ibrahim Asaad Aref

Submitted for the degree of doctor of Philosophy

February 2011

Abstract

This research uses a SystemC design methodology to model and design complex wireless communication systems, because in the recent years, the complexity of wireless communication systems has increased and the modelling and design of such systems has become inefficient and challenging. The most important aspect of modelling wireless communication systems is that system design choices may affect the communication behaviour and also communication design choices may impact on the system design.

Whilst, the SystemC modelling language shows great promise in the modelling of complex hardware/software systems, it still lacks a standard framework that supports modelling of wireless communication systems (particularly the use of wireless communication channels). SystemC lacks elements and components that can be used to express and simulate wireless systems. It does not support noise links natively. To fill this gap, this research proposes to extend the existing SystemC design methodology to include an efficient simulation of wireless systems. It proposes to achieve this by employing a system-level model of a noisy wireless communication channel, along with a small repertoire of standard components (which of course can be replaced on a per application basis).

Finally, to validate our developed methodology, a flocking behaviour system is selected as a demonstration (case study). This is a very complex system modelled based on the developed methodology and partitioned along different parameters. By applying our developed methodology to model this system as a case study, we can prove that incorporating and fixing the wireless channel, wireless protocol, noise or all of these elements early in the design methodology is very advantageous. The modelled system is introduced to simulate the behaviour of the particles (mobile units) that form a mobile ad-hoc communication network. Wireless communication between particles is addressed with two scenarios: the first is created using a wireless channel model to link each pair of particles, which means the wireless communication between particles is addressed using a Point-to-Point (P2P) channel; the other scenario is created using a shared channel (broadcast link).

Therefore, incorporating wireless features into existing SystemC design methodology, as done in this research, is a very important task, because by developing SystemC as a design tool to support wireless systems, hardware aspects, sofware parts and communication can be modelled, refined and validated simultaneously on the same platform, and the design space expanded into a two-dimensional design space comprising system and communication.

Declaration

This thesis presented a work that was carried at the University of Glasgow under the supervision of Dr. Khaled Elgaid and Dr. Fernando Rodríguez-Salazar, Department of Electronics and Electrical Engineering, during the period between March 2007 to December 2010. I declare that the work is entirely my own work and it has not been previously submitted for any other degree or qualification in any university.

Ibrahim Asaad Aref

Glasgow, February 2011

Acknowledgments

I would like to express my grateful acknowledgement to my supervisors Dr. Khaled Elgaid and Dr. Fernando Rodríguez-Salazar for their supervision and support. I wish to express my sincere thanks to the Electronics & Electrical Engineering, University of Glasgow for creating an environment that has been fabulous for research and fun. Also this work would not have been possible without the financial support from the Ministry of Education in Libya, I am grateful to them for this opportunity.

I am most appreciative of my father, mother, brothers and sister for their continuing support, as without their supports and wishes, I couldn't achieve what I have achieved today. Special thanks to my beloved wife for her patience and providing me comfort and my daughters and my son for giving me all the joy. Above all, I thank Allah almighty for his unlimited blessings.

Also I would like to thank the librarians for helping me to source the books and references that I needed to write my thesis.

Lastly, how can I forget to thank mates and colleagues, as without their help and friendly attitude, it was highly impossible to complete my thesis.

Ibrahim Aref

Glasgow, February 2011

Contents

List of Figures

List of Tables

List of Algorithms

List of Publications

The following publications have been based on the research work carried out during this Ph.D. project:

PAPER (1): I. A. Aref, N. A. Ahmed, F. Rodríguez-Salazar and K. Elgaid, "RTL-Level Modelling of an 8B/10B Encoder-Decoder using SystemC", in Proc. Fifth IEEE and IFIP International Conference on wireless and Optical communications Networks (WOCN_2008), Surabaya, Indonesia, May. 5–7, 2008.

PAPER (2): N. A. Ahmed, I. A. Aref, F. Rodríguez-Salazar and K. Elgaid, "Wireless Channel Model Based on SoC Design Methodology", in Proc. The Fourth International Conference on Systems and Networks Communications (ICSNC2009), September 20-25, 2009 - Porto, Portugal

PAPER (3): Ibrahim Aref, Nuredin Ahmed, Fernando Rodríguez-Salazar and Khaled Elgaid, "Wireless Extension Into Existing SystemC Design Methodology", in Proc. The 2nd IEEE International Conference on Computer Engineering and Technology (ICCET 2010), April 16 - 18, 2010, Chengdu, Sichuan, China.

PAPER (4): Ibrahim Aref, Nuredin Ahmed, Fernando Rodríguez-Salazar and Khaled Elgaid, "Measuring and Optimising Convergence and Stability in terms of System Construction in SystemC", in Proc. 17th IEEE International Conference and Workshops on Engineering of Computer-Based Systems. Oxford, UK, 22-26 March, 2010.

PAPER (5): Ibrahim Aref, Nuredin Ahmed, Fernando Rodríguez-Salazar and Khaled Elgaid, "Modeling of Flocking Behaviour System in SystemC", in Proc. The Sixth Advanced International Conference on Telecommunications (AICT 2010), May 9 - 15, 2010 - Barcelona, Spain.

PAPER (6): Nuredin Ahmed, Ibrahim Aref, Fernando Rodríguez-Salazar and Khaled Elgaid, "Network performance evaluation based on SoC design methodology", in Proc. The 7th International Symposium on Communication Systems Networks and Digital Signal Processing (CSNDSP), Newcastle upon Tyne, United Kingdom, 21-23 July 2010.

.

List of Symbols

μ	Mean and variance.
W	Work space.
P_k	Particle with index k.
εx	current error in x direction.
εy	current error in y direction.
K_p	proportional gain.
K_d	derivative gain.
$old\varepsilon_x$	error from the last cycle in x direction.
$old\varepsilon_y$	error from the last cycle in y direction.
$x_{acc.}$	current acceleration in x direction.
$y_{acc.}$	current acceleration in y direction.
N	Number of particles (Flock size).
L_d	desired distance.
R	Transmission rate.
V_x	Particle velocity in x-direction.
V_y	Particle velocity in y-direction.
p_i	Current position as vector (x,y).
r	Real position as vector (x,y).
C	Converging time.
ρ	Density of connections.
ec	Existing connections. = given conns.+randomly added cons.

gc	Given conns.= N (number of conns. in ring topology).
mc	Maximum number of connections.
e	Error percentage in the relative position.
P_r	Received power.
P_t	Transmitted power.
G_r	Receiver antenna gains.
G_t	Transmitter antenna gains.
d	Distance.
h_r	Antenna heights of receiver.
h_t	Antenna heights of transmitter.
P_0	power at a distance d_0
α	path loss exponent.
$\overline{PL}(d_0)$	the mean path loss in dB at distance d_0
χ	is a normally (Gaussian) distributed random variable.
σ	standard deviation

Glossary

AM	Amplitude Modulation.
AMS	Analog and Mixed Signal.
ARQ	Automatic Repeat reQuest.
ASK	Amplitude Shift Keying.
AUV	Autonomous Underwater Vehicles.
AWGN	Adaptive White Gaussian Noise.
BER	Bit Error Rate.
CLT	Central Limit Theorem.
CSMA	Carrier Sense Multiple Access.
DRHW	Dynamically reconfigurable hardware.
DSP	Digital Signal Processing.
DUT	Device Under Test.
FIFO	First In First Out.
FM	Frequency Modulation.
FPGA	Field-Programmable Gate Array.
FSK	Frequency Shift Keying.
GSM	Global System for Mobile Communications.
HDL	Hardware Description Language.
HDLC	High-Level Data Link Control.
IM	Intermodulation
IP	Intellectual Property

ISS Instruction Set Simulator.

JPEG Joint Photographic Experts Group.

LOS Line of Sight.

LSB Least Significant Bit.

MAC Media Access Control.

MSB Most Significant Bit.

MoC Model of Computation.

NS2 Network Simulator.

OFD MOrthogonal Frequency Division Multiplexing.

P2P Point-to-Point.

P2M Point-to-Multipoint.

PD ProportionalDerivative.

PDF Probability Density Function.

PLD Programmable Logic Device.

PLL Phased Locked Loop.

PM Phase modulation.

QAM Quadrature Amplitude Modulation.

RMS root mean square.

RMSE root mean square error.

RTL Register Transfer Level.

RTOS Real-Time Operating System.

SCNSL SystemC Network Simulation Library.

SDF Synchronous Data Flow.

SLD System Level Design.

SLDL System Level Design Language.

SMA Simple Moving Average.

SoC System-on-Chip.

SNR Signal-to-Noise Ratio.

TDMA Time Division Multiple Access.

TLM Transactional Level Modeling.

UART Universal Asynchronous Receiver/Transmitter.

UAV Unmanned Air Vehicles.

VHDL VHSIC Hardware Description Language.

VHSIC Very High Speed Integrated Circuits.

VOIP Voice Over IP.

WAN Wireless Area Network.

WCDMA Wideband Code Division Multiple Access.

WGN White Gaussian Noise.

WMN Wireless Mesh Network.

WSN Wireless Sensor Network.

Part I.

Introduction and Background

1. Introduction

The complexity of wireless communication systems has increased in recent years; the modelling and designing of such systems is becoming inefficient and very challenging because of many problems arising during the design process. It has become essential to improve currently used design methodologies. As a result of the problems shown, system designers are moving towards using System-on-Chip (SoC) design methodology. SoC is a complex integrated circuit that integrates the major functional elements of a large system into a single chip. It may integrate different types of components, such as digital elements, electronic elements or non-electrical parts (sensors), as well as software components [3, 4]. This design methodology can be used as a powerful modelling and simulation technique to design, model and verify complex wireless systems as well as address all aspects of the modelling process consistently and efficiently. SystemC [5], a hardware description language (HDL), is emerging as a suitable design and modelling language because it provides a consistent methodology for the design and refinement of complex digital systems [6]. This methodology is essential to manage complexity and enhance designer productivity. It allows the designer to view designs at different levels of abstraction, and in particular advocates evaluation of system performance early in the design cycle; it is useful in guiding the refinement process into lower levels of abstraction [4]. It is desirable to apply only one design methodology based on one modelling language to design complete complex systems, including any off-chip components.

SystemC design methodologies can be defined as approaches, sets of models, and procedures employed to convert the functional specifications of the target system to the system's lowest layer of abstraction [7]. It enables design and verification at the system level, independent of any detailed hardware and software implementation, as well as enabling co-verification with RTL design. It also builds the bridge between hard and software design [1, 8]. A number of efforts to extend SystemC design methodology to support a wide range of applications have already been undertaken, as in, for example, the forthcoming AMS (analogue and mixed signal) extension to the language [9, 10]; others also extend SystemC methodology for digital signal processing systems [11]. They present a SystemC-based solution supporting automatic design space exploration and automatic performance evaluation, as well as automatic system generation for mixed hardware/software solutions

mapped onto FPGA-based platforms; however, none have been specifically developed for the design of wireless communication systems, which are driving a large number of state-of-the-art developments in consumer products. Modelling and designing wireless communication systems is complicated and challenging, because during the system design exploration, system design options may affect communication behaviour and communication design options may impact system design [12]. For this reason it is important to introduce the integration of communication modelling into the design modelling at an early stage of system development. However, system designers are more focused on system design issues, whereas communication features are introduced by choosing traditional protocols; this makes the modelled system very difficult to optimise. To overcome this problem, powerful modelling and simulation techniques are required to address and manage complexity early in the design process. In the system modelling and design field, it is desirable and very advantageous to apply the same design methodology throughout the modelling of the system and communication aspects, to ensure that the whole system can be optimised and investigated easily and quickly [4, 13, 14].

In the literature, two types of modelling and simulation tools are employed to design a system: communication tools and system tools [15]. Communication simulation tools include Network Simulator (NS2) [16] and Opnet [17]; the problem is that these communication tools cannot be used for system design since they do not concurrently model tasks within each node in the communication system and do not provide a direct path to hardware/software synthesis [12]. Instead, system modelling tools, such as hardware description languages (HDLs), could be used to model communication, because they can model communication at least at high abstraction levels. As mentioned above, one HDL is SystemC [5], which has been traditionally used for system design and cannot be used to design complex wireless communication systems because it still lacks a standard framework that supports modelling of wireless communication systems (in particular the use of wireless communication channels). SystemC has certain elements, such as First-In First-Out (FIFOs), signals and semaphores [7]. The signal element behaves like a wire, which cannot be used to model a wireless system because no wireless features can be incorporated into this signal element, i.e., no noise model is supported by the SystemC language. Thus, SystemC lacks elements and components that can be used to express and simulate wireless systems. It does not support noise links natively. To fill this gap, this research proposes to extend existing SystemC design methodology to include an efficient simulation of wireless systems. It proposes to achieve this by employing a system-level model of a noisy wireless communication channel, along with a small repertoire of standard components (replaceable on a per application basis).

Thus, developing SystemC design methodology to support wireless features is a critical task for the designers, because SystemC provides a consistent methodology and a ho-

mogeneous design flow for the design and refinement of complex digital systems; it is also essential in managing complexity and enhancing designer productivity. After inserting wireless features into existing SystemC design methodology, the designers can very quickly investigate changes in the behaviour of the complex wireless communications system, allowing us to implement the system as we wish, view designs at different levels of abstraction and evaluate system performance early in the design cycle. These steps can all be achieved because SystemC is a unified environment, which means everything can be modelled in the same platform. However, the models developed of the target design could not be synthesizable, because the most accurate model that can be implemented using SystemC is RTL model. Hence to synthesis the target system, SystemC code representing hardware elements can be changed to HDL languages such as VHDL or Verilog and can then be synthesized. On the other hand, approaches that use different combinations of network simulation tools, such as NS2 [16] or Opnet [17], in the various stages of design flow require more than one tool to investigate the changes, so system blocks must be optimised again when they are integrated together; this is the main advantage of developing SystemC methodology to support wireless.

Finally, to validate our developed methodology, a flocking behaviour system [18] is selected as a demonstration. This is a very complex system modelled based on the developed methodology and partitioned along different parameters. By applying our developed methodology to model this system, we can prove that incorporating and fixing the wireless channel, wireless protocol, noise or all of these elements early in the design methodology is highly advantageous. The main point is the integration between computation, communication and SystemC methodology, i.e., introducing integrated communication modelling into the design modelling early in the system development. This allows us to investigate the system very easily and make changes quickly, because SystemC is a unified environment, which means everything is on the same platform. The modelled system is introduced to simulate the behaviour of particles (mobile units) that form a mobile ad-hoc communication network. Wireless communication between particles is addressed in two scenarios: the first is created using a wireless channel model to link each pair of particles [19, 20, 21], thus wireless communication between particles is addressed using a Point-to-Point (P2P) channel; the other scenario is created using a shared channel. Many protocols have been designed to handle access to the shared channel [22]. In this part of the research, two multiple access protocols are employed to accomplish communication between particles. The first protocol is Time Division Multiple Access (TDMA), which is classified as a channelization protocol. In this protocol, the available bandwidth of the channel is shared in time [22]. The second protocol is Carrier Sense Multiple Access (CSMA), which is classified as a random access protocol. It was developed to minimise the chance of collision and therefore increase system performance. It uses a contention-based approach to channel

access and does not require time synchronisation [23, 24].

Therefore incorporating wireless features into existing SystemC design methodology, as done in this research, is a very important task, because by developing SystemC as a design tool to support wireless features, hardware (HW), software (SW) and communication can be modelled, refined and validated simultaneously in same platform. That means, allow the designers to integrate system design options and communication choices at an early stage in the design flow flow. On the other hand, the design space expanded into a two dimensional design space, as shown in Figure (1.1). The vertical dimension addresses both the refinement of the system model and the optimisation of its algorithms; during this process the communication model is used both to drive architecture exploration and to verify that communication requirements are met. The horizontal dimension represents the design space of the communication model in which different topologies and parameters can be verified [15].

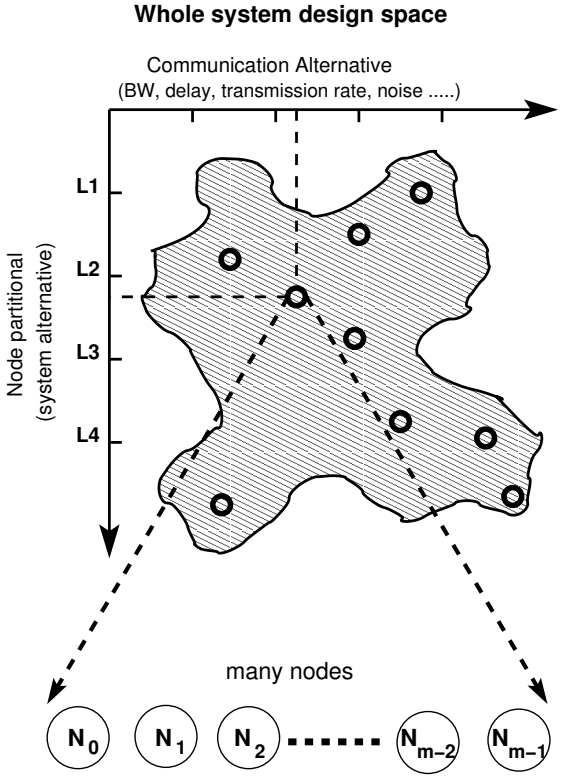


Figure 1.1.: Conventional methodology versus our developed methodology

1.1. Research Objectives

The aim of this research is to provide a wireless extension to existing SystemC design methodology. To achieve this target, we propose the following three phases:

1. Design a digital wireless communication channel. This represents the core of any wireless communication system. It is needed to link network nodes in wireless systems.

2. Construct system-level models for a small set of standard components that are typically required to implement wireless communication system elements, such as PLL (Phased Locked Loop), 8B/10B Encoder/Decoder, MAC (Media Access Control), communication protocols such as HDLC (High-Level Data Link Control), modulation techniques, etc.

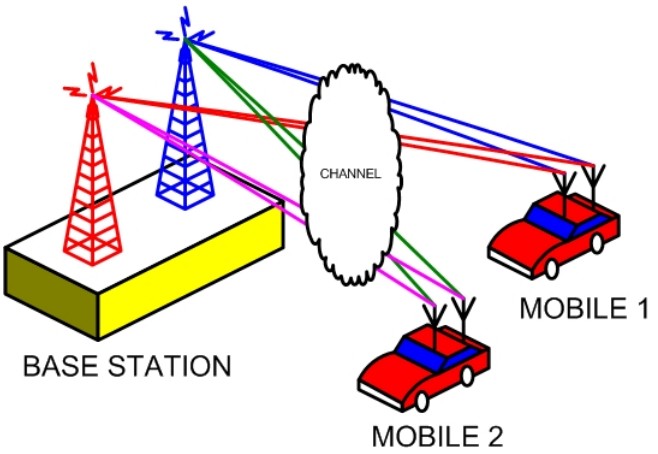
3. Create a demonstration in order to validate the methodology by developing a small application and/or test case, such as a flocking behaviour system, fully connected Wireless Mesh Networks (WMNs) or Wireless Sensor Networks (WSNs). We can design such systems as test cases using the elements constructed for the above points.

1.2. Application Domains

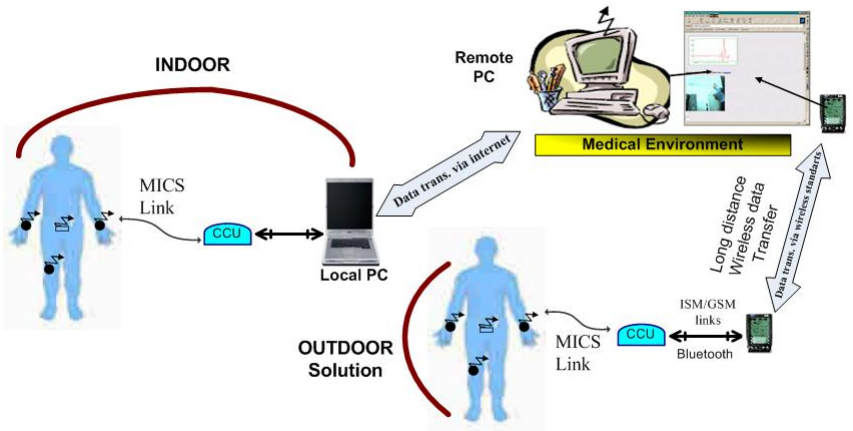
The extended SystemC design methodology can be applied in multiple application domains. It is developed to support different application domains in wireless communications fields. The main focus is on modeling of the heterogeneous wireless communication systems such as unmanned aerial vehicles (UAV), wireless sensor networks (WSN), cellular telephones, etc. as shown in Figure(1.2).

1.3. Thesis Organisation

This PhD thesis comprises three parts. The first part, which is the introduction and background, has two chapters, including this one. Chapter 2 provides the necessary background, in considerable detail, on conventional SystemC design methodology, as well as channel modelling methods and flocking behaviour algorithms. The second part of the thesis comprises Chapters 3 to 5, and focuses on wireless methodology. Chapter 3 provides RTL-level modelling of an 8B/10B Encoder-Decoder using SystemC. Chapter 4 describes in detail how to develop existing SystemC design methodology by incorporating wireless features and then how to employ this methodology to model wireless communication systems. Chapter 5 presents a new method to model and simulate a noisy digital wireless communication channel with a preset Bit-Error-Rate (BER) for P2P and Point-to-Multipoint (P2M) platforms based on SystemC. The flocking behaviour system case study



(A) Taxi mobile



(B) Wireless medical monitoring [25]

Figure 1.2.: Some examples of heterogeneous wireless communication systems

is developed to validate our wireless methodology; this is presented in part three, which comprises five chapters. Chapter 6 presents modelling of a flocking behaviour system based on our developed methodology, where wireless communication between particles is addressed using a P2P channel. Chapter 7 investigates the effects of wireless features on the modelled system by incorporating the wireless channel model. In Chapter 8, we use the flocking behaviour system modelled in Chapter 6 to show how the stability of the system and converging point are measured and optimised in terms of system construction, using some important concepts of graph theory. Chapter 9 presents communication modelling based on a shared channel (broadcast link). Finally, we conclude the thesis with conclusions and possible future work in Chapter 10.

2. Background

As mentioned in Chapter One, this research studies how wireless features, including the efficient simulation of wireless systems, can be incorporated into existing SystemC design methodology. In this chapter we provide background information about the main components investigated in order to achieve the research target. This information is necessary in order to understand the material found in subsequent chapters. In the first part of this chapter we describe existing SystemC design methodology; the second part illustrates how a communication channel model can be developed. Finally, we explain the concept and main rules of the flocking behaviour system constructed as a case study to validate the proposed wireless methodology.

2.1. System Design using SystemC Methodology

System design methodologies can be understood as approaches, sets of models, and procedures employed to convert the functional specifications of the target system to the system's lowest layer of abstraction [3, 7]. One of the emerging design methodologies, SystemC methodology, is very flexible in terms of the design and refinement of complex digital systems. This is essential for managing complexity and enhancing designer productivity [26].

Since our target in this research is to develop SystemC design methodology to support wireless systems, therefore we need to explore the existing SystemC methodology in order to make this extension. This part begins with an overview of the System-Level Design Language (SLDL). Some background on the SystemC language is presented in Section 2.1.2. Section 2.1.3 describes conventional SystemC methodology and the roles of the levels of abstraction that are widely used in the system design and modelling field. Section 2.1.4 presents the system design flow of SystemC methodology.

2.1.1. System Level Design (SLD)

System-level design (SLD) is defined as the design process of the entire system, which is constructed based on several components. The specifications of the target system are presented in terms of functionality [27]. An SLD approach can be employed to produce system implementation from system functionality and specifications [1]. It can be involved to define the target system to verify system functionality and then to find an optimal solution by comparing design alternatives. The software part of the system is integrated into a SoC [26, 28]. Moreover, with the increasing complexity of hardware and software, system-level design can be used to help designers reduce system design complexity by defining a number of intermediate models; this approach can be used to reduce the time required to market and reduce design costs [14, 29].

Several system-level design languages (SLDLs) have recently become available; these can be immediately employed by system designers in order to model HW/SW designs at system level. SystemC [30, 4], which is based on the C/C++ language, is an example. These languages allow designers to describe both HW and SW aspects of the design using the same modelling language [28].

2.1.2. SystemC Language Overview

SystemC is a system design language that provides designers with basic mechanisms, such as channels, interfaces and events, to model systems, communication and synchronisation styles in system designs at various levels of abstraction [13, 8, 31]. The software content of the system can be written in C++, without the need for additional constructs. The power of SystemC is that it can be used as a common language by system designers, software engineers and hardware designers [32, 33].

The advantages of SystemC include the ability for hardware/software co-design, the ability to exchange Intellectual Property (IP) easily and effectively, the establishment of a common design environment consisting of C++ libraries, models and tools, and the ability to reuse test benches across multiple levels of design abstraction [26, 34]. Moreover, the design can be incrementally refined with the addition of hardware and timing constructs to arrive at the final target architecture. SystemC ensures a smooth flow in capturing design details at multiple abstraction levels, starting with an algorithmic-level implementation used to verify the functionality of the system, up to a cycle-accurate design. The major hardware-oriented features included within the SystemC library are [6, 26, 35]:

- Time model.

- Hardware data types.
- Module hierarchy to manage structure and connectivity.
- Communication management between concurrent units of execution.
- Concurrency model.

Moreover, there are several approaches to designing and simulating behavioural modelling of embedded systems by using software tools such as Matlab [36] and Simulink [37]. These tools are flexible and easy to learn but, being commercial products, are more expensive. Alternatively, Verilog, VHDL and other HDLs model simulate digital and electronic systems. Also there is Handel-C [38], a language based on ANSI-C, employed to simulate parallel programs, channel communications and extended with concepts for timing and concurrency [39].

However, these approaches do not deal as effectively with the increasing design complexity of these systems as SystemC [8], which allows early and rapid verification of both the hardware and software aspects of the entire System. It promotes early software development, so the software-part is not necessary on the critical-path. SystemC also forwards design reuse and automation [40]. There are more SystemC features, but here we have tried to provide a brief sketch by listing the most important aspects of this library. Beyond that, there are good references for SystemC in [5, 6, 26, 28, 8, 31, 33, 41], which explain the complete SystemC functionality.

2.1.3. SystemC Design Methodology

The design and verification of complex digital systems requires powerful modelling and simulation means to address all aspects consistently and efficiently. However, system designers face a big issue: how to manage increasing system design complexity. Design methodology is the key for managing the complexity of the design flow, especially at the system level [4, 26, 42]. As mentioned above, the primary goal of SLD is to make the design and modelling processes of such systems easier for designers to manage. One essential aspect of an SLDL is the ability to support modelling of the system at multiple layers of abstraction, represented by a number of intermediate models. These layers are defined in order to reduce the complexity of the system design process and to evaluate system performance early in the design cycle [13, 43]. Hence the whole design can be separated into several small tasks, with each task represented by one abstraction level. Finally, each model can be simulated and the result of the simulation can be independently validated [7, 44].

The works in [1, 45] introduce a system modelling graph in two dimensions: computation, represented by the X-axis, and communication, represented by the Y-axis, as shown in Figure (2.1). This graph is mainly used to represent the system design domain and illustrates how much timing information is included in the computation and communication aspects. Hence, all the models defined above have a specific position in this graph, based on the timing information. Both axes of the graph move from very little timing information to complete timing information, which means from untimed to approximate timed to cycle timed. To model a system, the design should start from an untimed model that represents the pure functionality of the design without any implementation details, and then go through some intermediate models before arriving at the final model, which is an RTL model, as indicated in Figure (2.1).

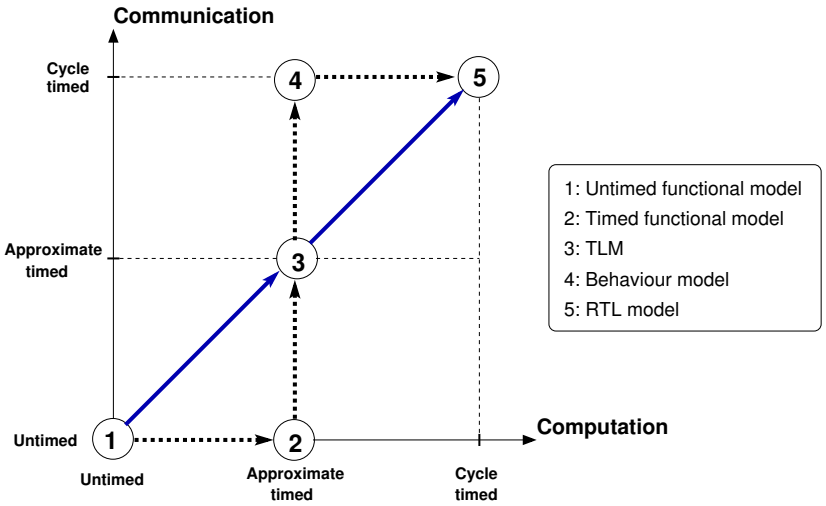


Figure 2.1.: System modelling graph [1]

The system modelling graph indicated above does not necessarily need to be represented in two dimensions. We can extend it to three dimensions if the models described above are mapped onto a 3D graph, as illustrated in Figure (2.2). In this case, we have three axes. The first axis is timing, representing how much timing information is added, from an untimed level to a cycle-timed level. The second axis is architecture, representing how much the level of architecture complexity is in the current model. The third axis is communication, representing the level of communication in the target system. At the origin point of the graph, the functional model is located and is known as the highest abstract timing model, which means no timing information is known at all. Functional means nothing is known about architecture, only about implementation. When designing a system we always start from a functional model and move towards a final model, an RTL model. As shown in Figure (2.2), when the level of abstraction goes into the origin point, the target model becomes more abstract. On the other hand, information is opposite of abstraction, which means less information is a higher abstraction level, and adding more

information decreases the abstraction level. Depending on the information added, we can establish more abstraction levels. To design a system, we move between the 3-dimensional axes until the target model is determined, which should be located at the lowest abstraction levels of timing, architecture and communication.

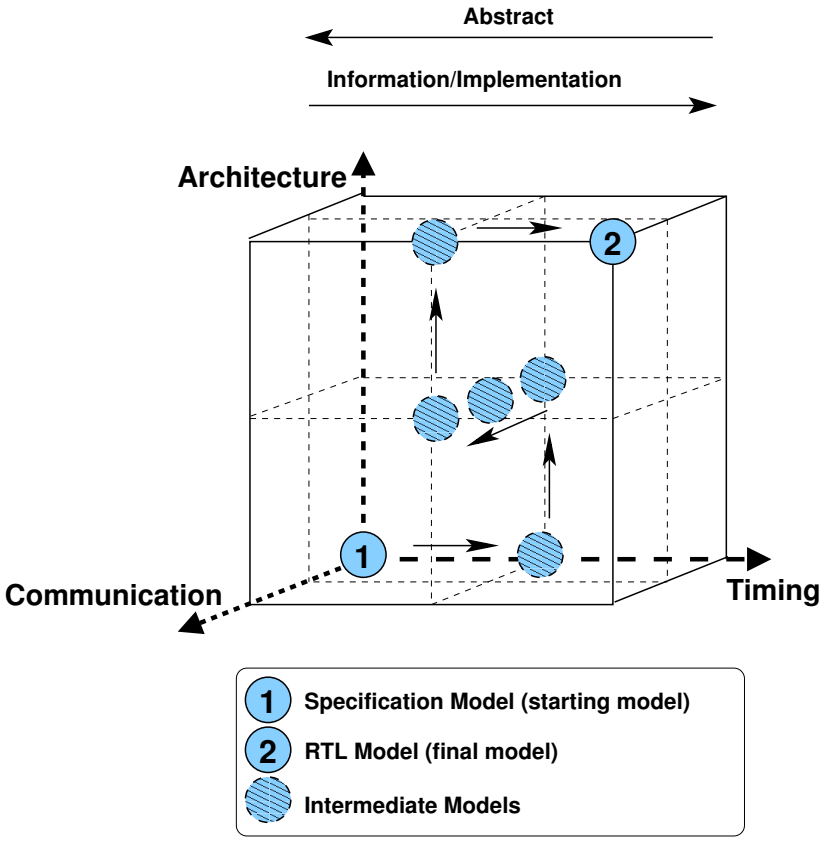


Figure 2.2.: Abstraction Levels in 3D

As shown in Figure (2.2), the functional model is located at the origin point (position-1), RTL is located at position-2 and TLM and behaviour levels are located between these two points. Those intermediate models represent the abstraction levels. But these are not exclusive: there are some levels in between. It is not one model and other but it is a global refinement along these abstraction models, which means we can increase or decrease the number of abstraction levels: this means there is no specific number of abstraction levels. We can create a huge number of levels by adding more timing, architecture and communication information.

SystemC as an SLDL offers high abstraction level timing modelling through transaction level modelling (TLM), causing substantially increasing simulation speeds compared to conventional RTL simulation, and reducing system design complexity [34, 43, 46]. This saves time during debugging and allows for more efficient architecture exploration. Moreover, SystemC provides designers with a top down design methodology, which means that

the system level is said to be at the highest or top level of abstraction, and the model is refined as it moves down to lower levels, as shown in Figure (2.3) [1, 8, 47]. These intermediate levels divide the entire design into small design tasks; each small task has a specific design objective. Since each task can be modelled and simulated, the result of each model can be independently validated; the system design can be refined and validated at each level in terms of architecture. The next sections provide a short description of SystemC modelling methodology abstraction levels [43].

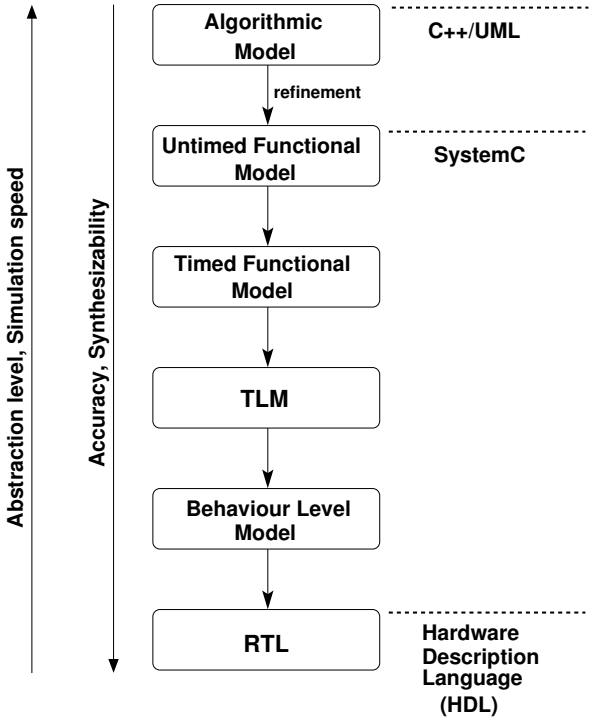


Figure 2.3.: SystemC design methodology [1]

1- Algorithmic Model

An algorithmic model can be employed to develop an idea for a target design. The functionality of the design can be confirmed and the output may be created for verification purposes in later system stages [48]. This model has no timing, size or cost constraints information.

2- Untimed Functional Model

An untimed functional model is referred to as a specification model. It must be a pure, functionally accurate model of the target system, without any implementation details. At this level a system model is similar to an executable specification. As the name implies,

this model is an untimed model in terms of both computation and communication, which means no timing is incorporated into this type of model and there is no reference to any architectural details of the system [7, 41, 49]. Communication in this model is point-to-point and implemented with SystemC primitive channels such as first in, first out (FIFO), without implementation of communication protocols or shared communication links, while synchronisation between modules is usually implicit with blocking FIFO buffers. When this model is simulated, the functional results can be verified and validated [1, 50].

3- Timed Functional Model

Here, the untimed model is refined into a timed model by partitioning the design functionality. The partitioned functionality is mapped onto components and the execution timing of the processes are added to these components in order to reflect the timing constraints of the design specification and also to estimate processing delays for the target architecture, which means the latencies are modelled [41, 50]. Hence, this model is approximate-timed in terms of computation. On the other hand, communication is still modelled at an abstract level and is still point-to-point. The system components communicate through the self-synchronising channels mentioned above, so this model is untimed in terms of communication. At this level, this model can be employed to achieve early hardware-software tradeoff analysis [13, 7].

4- Transaction-level Model (TLM)

In a TLM model, the communication of the target system is separated from the behaviour and refined from a high abstraction level to a bus model. The behaviour is represented by modules and the communication busses are implemented using function calls related to communication, allowing for an easier architecture exploration [7, 45]. The time spent during data exchange is represented by wait statements and is inserted into the bus model (channel). The computational element is still approximate-timed, as indicated in the previous model. Both computation and communication are approximately timed, which means the system design at this level delivers high-speed simulation and presents a virtual prototype of the final design [46, 50].

5- Behaviour-level Model

In this model, the communication bus model is refined into more detailed implementations, and the communication protocols of the system bus are incorporated into the processing

elements. The communication busses are represented by wires and pins added to the processing elements, so the communication protocols are inlined into processing elements (pin accurate interfaces) [50]. Since computation is still approximately timed, as indicated in the previous model, the main difference here is that communication is refined to more accurate model (cycle-accurate). By using this model we can get better performance analysis of the design, but at the cost of decreased simulation speed. This model can also be employed for fine-tuning the system and refining the implementation [7].

6- Register-Transfer Level Model (RTL)

The RTL model is defined as the most accurate model that can be implemented using SystemC; to date, there are no tools available on the market that can be used for implementation beyond RTL in SystemC. In this model, the computation behaviour and communication channels of the target design are defined explicitly (both are cycle-accurate) [7, 49]. Moreover, a SystemC code representing hardware elements is changed to HDL and can then be synthesized. On the other hand, a SystemC code representing software is changed to the desired programming language [13].

2.1.4. System Design Flow with SystemC

The SystemC language enables designers to model and design whole systems (HW and SW components) at different abstraction levels, using one descriptive language. In SystemC design flow the target design moves from the highest abstraction level (system specification level) to the lowest (synthesisable hardware description), in small refinement steps [29, 33]. By using this approach, designers can more easily implement changes in the design at each level and then verify the model. The goal of this design approach is to avoid some of the disadvantages of traditional design flows as well as achieve early modelling of the entire system, to allow designers to save time, evaluate system performance early in the design cycle and save design costs [47]. The design flow of the whole system is fragmented into several models, defined in the previous section. Each model represents the abstraction level at which a current implementation is modelled. All phases in SystemC design flow are depicted in Figure (2.4) [44, 49].

According on our description above, we can say that the design flow of SystemC design methodology is based on the methodology stages described above. As shown in Figure (2.4), the first step of the SystemC design flow is to capture functional specifications in C++ abstraction. By the end of this step, the complete system functionality is captured and validated. Some of the system specifications that can be introduced in this stage are

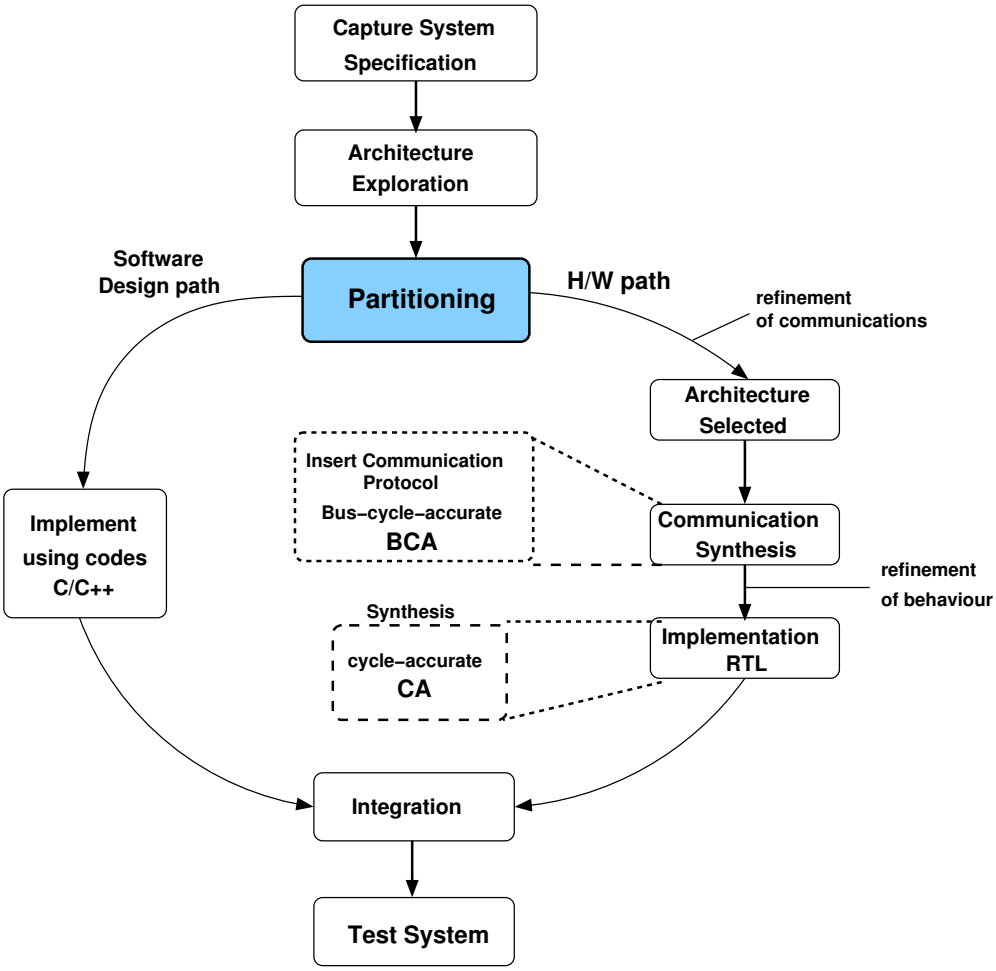


Figure 2.4.: SystemC design flow

concurrency, replacing native C++ data types to bit accurate SystemC data types, inserting computation delay, partitioning target design and allocating system functionality into components.

After capturing the specifications and refining the untimed functional model to a timed functional SystemC model, the next step is architecture exploration. The main purpose in this phase is to allocate system functionality into components. The system architecture is derived from the specifications. We can develop more than one option/design and compare them. We must estimate a metrics such as area, timing, power consumption etc that will be employed to investigate system performance for these options and then select the optimum design. The hardware/software partitioning then takes place. In this phase, the components selected as S/W will be implemented using S/W code and the other components will be exported to SystemC language to be implemented as H/W components [1, 14]. Next, further refinement can be done in order to get bus-accurate description and then pin-accurate description.

Several approaches to designing and simulating behavioural modelling of embedded sys-

tems based on existing SystemC design methodology have been proposed and published. For example, the behavioural modelling of analogue-to-digital converters (ADCs) based on SystemC is presented in [51]. The authors design a 12 bit pipelined ADC in order to compare the performance of their SystemC framework with Verilog, and report the results. Also, the results presented in [52] emphasize the facility to describe hardware using SystemC, which provides a faster simulation time compared to VHDL. Moreover, SystemC is more efficient in validating hardware than VHDL, and potentially enables a much wider community to design hardware. The author of [40] compares traditional design flow and SystemC design flow, indicating that SystemC flow starts with an executable specification in SystemC as a base for hardware and software design. The advantage to this is that the hardware and software models exist in a homogeneous environment. This permits gradual refinement of the hardware design, without any language barrier.

The authors in [53] investigate the capabilities of existing SystemC methodology by designing and simulating a complex digital system starting from the highest abstraction level; they demonstrate the main features of SystemC, such as supporting timed behavior, hierarchy, concurrency and the creation of fast executable specifications of the target design. Also, the authors examine their proposed design flow by applying it to capture and validate the target design at the system level. Moreover, as an example of a complex digital system used to test their design flow, the authors select a complex Iterative Turbo Decoder algorithm as a prototype system. To support their results, the decoder behavior model at the system level using SystemC is compared to an RTL model designed using Verilog 2001, and the final results show that the model implemented using SystemC was executed approximately 10 times faster than the corresponding models designed at RTL level using Verilog 2001. Finally, the authors in [53] successfully prove the effectiveness of SystemC in exploring a huge system design space.

Similarly, the authors in [47] examine how we can use SystemC design methodology to model and design an effective system with high performance. This includes evaluating all the system modelling stages in the design flow, from functional level to RTL level, and determining how the methodology can be used to improve system design. As part of their investigation, a Joint Photographic Experts Group (JPEG) encoder and universal asynchronous receiver/transmitter (UART) were successfully designed using SystemC design flow. These implemented models show how SystemC overcomes issues presented by different design methodologies. More over, this work demonstrates that SystemC is worth investigating as a suitable HDL to model and design high performance systems.

In order to facilitate an SLD approach, [54] defines some additional refinement of SystemC methodology that can be employed to help designers manage the complexity of the different embedded systems, from the highest level of abstraction to the lowest. As a case study,

the authors design a digital camera to validate their proposed top down design methodology. In [55, 56], the authors present system-level support for the design of dynamically reconfigurable hardware (DRHW) based on SystemC, and also introduce many configuration management approaches that can be used to reduce the impact of configuration overhead. Moreover, they have developed an estimation method for system partitioning and DRHW modelling. The main goal of their approach is to help system designers easily evaluate the effect of changing some components from fixed hardware implementation to DRHW. The support is applied in a Wideband Code Division Multiple Access (WCDMA) case study.

2.1.5. Extending SystemC Methodology to Support Wireless Features

As previously mentioned, this research studies how wireless features can be incorporated into existing SystemC design methodology, including the efficient simulation of wireless systems. At present, the SystemC modelling language lacks a standard framework that supports the modelling of wireless communication systems (in particular the use of wireless communication channels), because it is designed for modelling digital systems. It is important to mention that, to the best of our knowledge, SystemC design methodology has not previously been extended to support wireless features. However, in recent years some researchers have proposed to extend SystemC to model and design analogue and mixed-signal systems, such as in [10, 57, 58, 59, 60], and other researchers have extended the capabilities of SystemC to model radio frequency parts [61]. For instance, the authors in [10] propose an extension to the capabilities of SystemC to allow modelling of analogue and mixed-signal systems, because at the time of its writing SystemC still lacked elements that could be used to model and design continuous-time systems. To fill this gap, team called SystemC-AMS [9] was formed to develop SystemC to support analogue and mixed-signal systems (AMS), hence the name SystemC-AMS. They describe a design methodology to model such systems and, by using this methodology, analogue signal processing applications can be efficiently supported without the need for sophisticated continuous-time simulators.

The researchers in [58] define design objectives of analogue extensions to SystemC with respect to requirements related to different application domains. They propose that the development would involve three phases, with new capabilities being added in each phase. Moreover, they present the first version of the SystemC-AMS functional specification in [57], containing the first elements of phase 1 of the extension of SystemC framework to support analogue and mixed-signal systems. They also show how integration between

the analogue elements in the existing SystemC 2.0 environment was achieved. Another approach is proposed in [33] by the same group, the SystemC-AMS group, to extend the existing SystemC environment to support mixed discrete-continuous systems by implementing a synchronous dataflow (SDF) model of computation (MoC). They mainly use the SDF MoC to embed continuous-time behaviour in SDF modules and to support synchronisation with the SystemC environment. They also present an overview of the architecture and syntax of the proposed extensions including examples with results. The authors in [61] propose an extension of the capabilities of the SystemC environment to allow modelling RF systems in SystemC-WMS[62], which is based on a wave mixed signal class library created to extend the existing SystemC environment to allow for the design and modelling of complex systems comprising heterogeneous analogue parts. They model analogue RF parts at the system level by using only their specifications. Also, by using the proposed methodology, a complete Bluetooth transceiver system consisting of digital and analogue elements was designed and simulated in order to validate their methodology. None of the works mentioned above extend the SystemC environment to model wireless communication systems, indicating the uniqueness of the present research.

On the other hand, several approaches targeting integration between system design modelling and communication aspects exist in literature. However, most of these researches propose a framework supporting inter-operation of different tools in order to model the system and communication aspects, meaning they have not incorporated the whole design process into a homogeneous design methodology based on a single language. The authors in [63] present a co-design methodology to join system alternatives with network features. In this approach, the HW/SW part of the system is modelled using SystemC and the network part is implemented using a network simulator (NS2) [16]. Furthermore, the authors modify the simulator kernels in order to share a common simulation time. In this methodology, the partition phase is done through two steps: the functionalities of the whole system are allocated between system aspect and network components, and the latter functionalities are assigned to either HW or SW. In order to validate their methodology, the authors successfully apply it to model the fast path of IPv4 routing, allowing it to explore different solutions based on system and network configurations.

In [64], a methodology is explored for modelling an embedded system and its interaction with a network. It joins SystemC and NS2 to model system and network aspects respectively. The interface between these modelling tools is achieved through a shared memory queue. The authors also apply their proposed methodology to two case studies: the modelling of a network device and the verification of two cooperating embedded systems. In [65], a modelling and simulation methodology is used to design networked systems based on a timing accurate integration of SystemC as a hardward/software modelling tool and NS-2 as a network simulation tool. The integration between these tools is implemented

at the level of the simulation kernels in order to provide the highest efficiency level; this integration also supports a timing accurate synchronisation of these simulation tools. The efficiency of the proposed methodology design is tested by modelling an 802.11 MAC layer. The results of this case study indicate the high efficiency of the proposed methodology.

In [66], the authors develop a methodology to model system/network aspects of a heterogeneous network. The system is constructed from a mobile phone, a wireless sensor network (WSN), remote hosts and a wireless area network (WAN). The mobile phone is used as a gateway between the WSN and the WAN. It exchanges data with the WSN and data/voice with remote hosts through the WAN. The gateway is modelled at the system level with SystemC, while the WAN is simulated with NS2. The WSN has been completely modelled in SystemC while the WAN has been modelled with NS2. Moreover, to achieve cycle accurate execution of the RTOS and the application software, HW/SW partitioning has been applied to the initial model of the gateway and an instruction set simulator (ISS) of the ARM processor has been employed. At a system level scenario, the system results indicate that the total elapsed time depends on the WSN size. The experiment shows that with ten nodes the simulation is faster than real-time. Also, [67] and [15] integrate TLM modelling with the system/network to model network embedded systems. In this approach, the authors use network configurations to drive architecture exploration and to validate the system model after each refinement step. They provide a general criterion in order to map whole system functionalities to system alternative and network aspect. To validate their proposed methodology, a Voice-over-IP client is modelled as case study.

The authors of [68] address the problem of simulating the heterogeneous networked embedded systems which assist to construct reliable, secure and scalable applications. They discuss and illustrate how to combine different modelling tools to provide different modelling and simulation alternatives for the design of networked embedded systems. The problem is investigated theoretically and practically by applying their approach to model a real application obtained from a European project, consisting of wireless sensor nodes interacting with traditional networks through a gateway. The authors mention that the use of SystemC is a possible solution to designing and simulating network embedded systems, but the main drawback of this approach is that the designer must implement the network behaviours since, the communication aspect cannot be simulated because it is not supported by SystemC, this means that there is no available library in SystemC to support network behaviours.

Hence, one can say that for any changes in the above approaches and methodologies, more than one tool must be used to investigate the change, because the system blocks must be re-optimised once they are integrated. The wireless methodology developed in

this research relies on just one modelling language, SystemC, so we can investigate the system's behavioural changes very quickly, because SystemC is a unified environment, which means every thing can be modelled in the same place. This is the main advantage of our developed SystemC methodology over the approaches mentioned above. On the other hand, one of the few studies using a unified environment throughout the modelling of the system aspects and networks is found in [12, 69]. The authors have exploited the SystemC language to build a System/Network simulator called SystemC Network Simulation Library (SCNSL). This library allows for model network scenarios in which different kinds of nodes, or nodes described at different abstraction levels, interact. Here the authors use SystemC as a unique tool allowing them to model, validate and refine the system and network in the same environment, but this approach is different from the present research.

2.2. Radio Communication System

In a radio communication system, communication is achieved and signals are transmitted based on radio (radiated emission) [70]. Radio emission is classified depending on the characteristics of the signal, such as modulation technique (Amplitude Modulation (AM), Frequency Modulation (FM) or Phase Modulation (PM)), the nature of the modulating signal, transmitted data type, licensing restrictions and radio bandwidth [71, 72, 73]. The essential elements of any radio communication system are: a transmitter, a transmission medium and a receiver [72]. Each transmitter and receiver comprises an antenna and appropriate terminal equipment that can be used as an input or output device, such as a microphone at the transmitter and a speaker at the receiver [70, 74]. The transmission medium is a wireless channel that is subject to various types of noise and other wireless features [72, 74]. So we can say that the wireless channel represents the core of any wireless communication system; in this research its model is used as a building block for the development of SystemC methodology to support wireless communication. As a result of noise effect, the behaviour of the wireless channel becomes unreliable and the state of the channel may change frequently. This random behaviour of wireless communication channels make communication over such a channel very complicated [75]. To provide clear background information, this section reviewed the importance of accurate channel models and investigated the role these models have on designing an efficient communication system. We provided an overview of wireless channel modelling as well as its key characteristics. We also reviewed some of the wireless channel models found in the literature.

2.2.1. Wireless Channel

Channel modelling in general is an important research topic in wireless communication. There are many communication channel models that have already been successfully implemented to simulate different environments. To define a channel model, we can say that a channel can be modelled physically by trying to calculate the physical processes which modify the transmitted signal [76, 77]. The channel model represents the transmission channel, which is the medium between the transmitting antenna and the receiving antenna [71]. The simplest wireless communication system model can be represented by a transmitter, a wireless channel and a receiver, as shown in Figure (2.5). In such a system, the signal travels through the wireless channel from the transmitter station to the receiving station, using the transmitter antenna, and is received together with interference and noise by the receiver antenna [78, 79]. As illustrated in Figure (2.5), interference and some noise affect the transmitted signals. Interference is produced from other radio transmitters or other electrical equipment that emits radio frequency energy. Noise types are inserted or incorporated into the signal as it travels in the transmission medium (the atmosphere), and some noise is added from thermal and other effects inside the receiver. In the last stage, at the receiving side, the signal is filtered by a band pass filter in order to reach the target signal [79, 80].

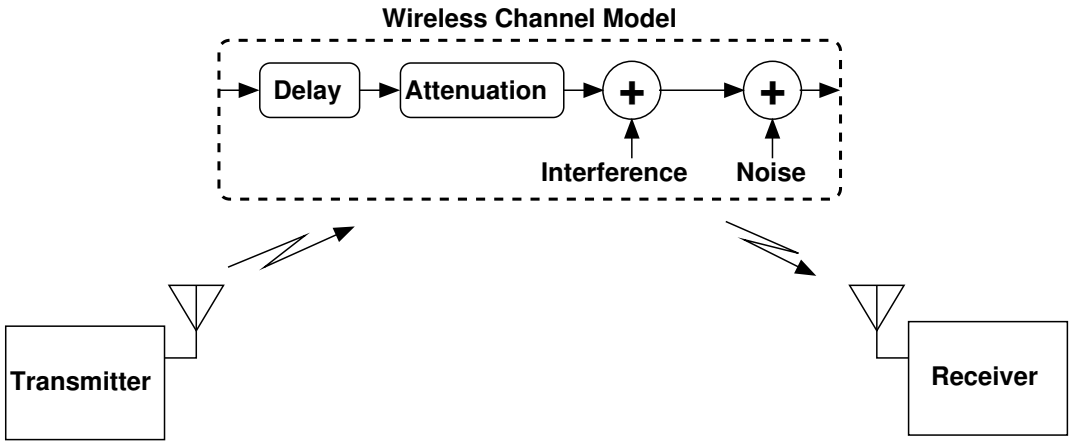


Figure 2.5.: A simple wireless communication system model

In the case of wireless communications systems, which are more relevant to our research, wireless channels are classified according to their propagation environment: urban, suburban, indoor, underwater or orbital; these propagation environments differ in various ways [75]. Under these conditions, the performance of the wireless communication systems mainly depends on the wireless channel behaviour, because, for example, the communication path between the sender station and the receiver can change from simple line-of-sight

to one that is drastically obstructed by buildings, foliage and mountains. Hence, the modelling and design of such a channel becomes the most difficult part of the wireless system [75, 81]. The wireless channel can be modelled by calculating the reflection off every object in the environment. A sequence of random numbers might also be added to simulate external interference and/or electronic noise in the receiver. Also, the communication channel can be modelled statistically, or statistical and physical modelling can be combined. In wireless communications the channel is often modelled by a random attenuation (known as fading) of the transmitted signal, followed by additive noise [75, 82].

2.2.2. Characteristics of Wireless Channels

In radio communication systems, radio waves are propagated outwards from a transmitting antenna to a receiving antenna, but some factors can seriously impact the propagations of these waves. Reflection, diffraction and scattering represent the three basic factors that impact mobile communication systems [83, 84].

- **Reflection:** This arises when propagating electromagnetic waves are incident on an object with larger dimensions than the incident wave's wavelength. For example, the reflection mechanism can occur at the surface of the Earth, as shown in Figure (2.6), or at walls [83, 85].

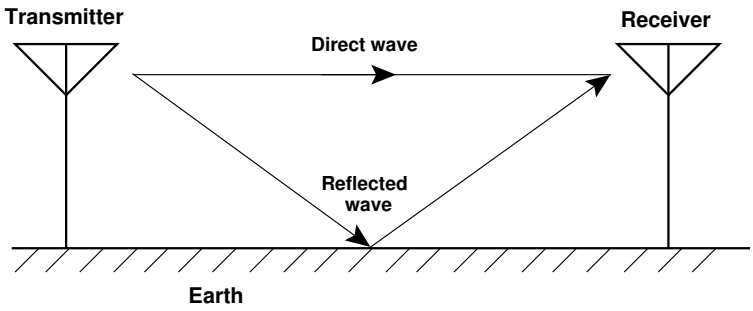


Figure 2.6.: Reflection mechanism at the surface of the Earth

- **Diffraction:** This occurs when a direct radio path from the transmitter antenna to the receiver antenna is obstructed by a sharp object, as shown in Figure (2.7), and, as a result of this, secondary waves are generated behind and around the obstructing object [86].

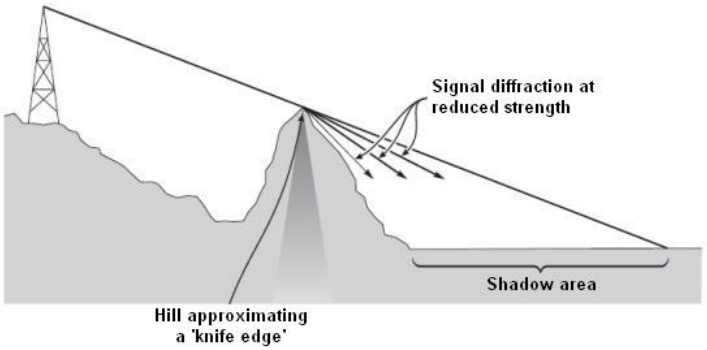


Figure 2.7.: Diffraction phenomenon

- **Scattering:** Scattering arises when the direct radio path of the wave propagation consists of objects with a non-regular shape that have dimensions smaller than the wavelength of the transmitted wave. As a result, the energy of the transmitting wave is redirected in different directions around the obstacle, as shown in Figure (2.8) [84, 85].

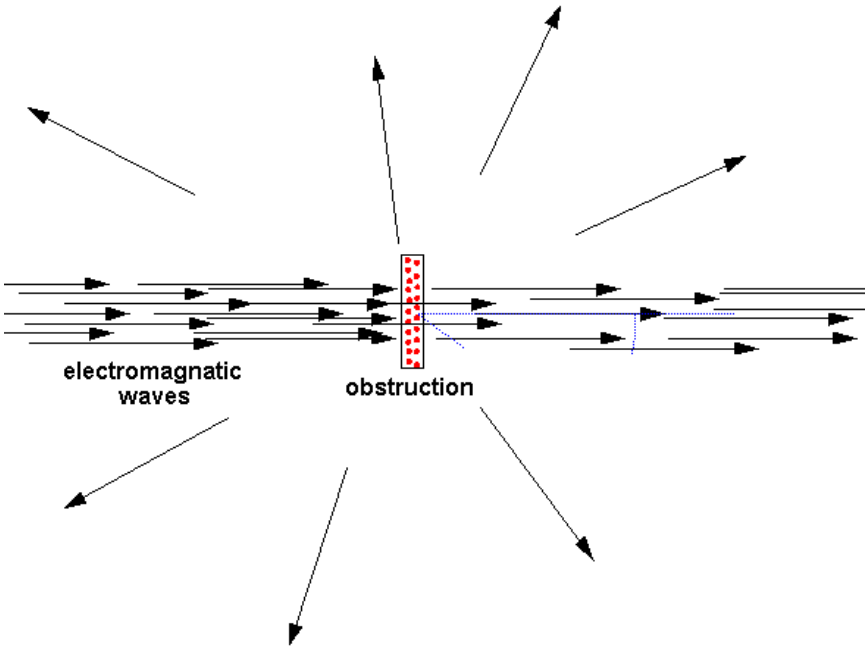


Figure 2.8.: Scattering

In this section we cover what happens to the signal as it travels from the transmitter to the receivers. The wireless signal that travels from the transmitter antenna to the receiver antenna is affected by some of the factors mentioned previously, so the behaviour of the wireless channel can be characterised by three independent phenomena. These are: path loss variation with distance, shadowing and multipath fading [71, 85]. These characteristics depict the variations of the channel strength over time domain and frequency domain.

The wireless channel variations can be classified into two types: large scale channel variation and small scale channel variation, as shown in Figure (2.9) [74].

- **Large-scale fading:** This refers to variations due to path loss of the signal as a function of distance and shadowing by large objects (obstacles), such as buildings and hills. This variation occurs when the receiver moves a relatively large distance from the transmitter, so the signal strength gradually decreases. Large-scale fading is typically frequency independent [73, 87].

- **Small-scale fading:** This happens when a receiver moves a short distance. The signal strength may vary rapidly due to the constructive and destructive interference of the multiple path propagation effects that occur over very short distances. This phenomenon is frequency dependent [73, 87].

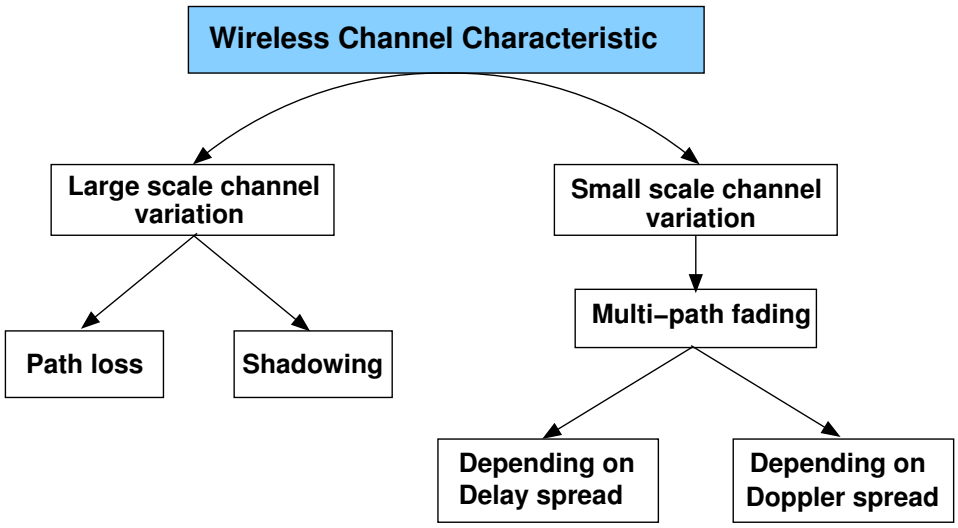


Figure 2.9.: Wireless channel characteristics

a- Path Loss in Propagation

As a signal travels from the transmitter to the receivers, it goes through several sources of attenuation. In a free space, the signal will lose power; as a signal leaves the transmitter and moves towards the receiver, its power will drop, and the dropping ratio depends much on the distance and the medium in which the signal travels. This means that the greater the distance, the more power will drop [87]. In free space, assuming for simplicity that the transmitted signal propagates uniformly in all directions, the following analysis holds. We assume the transmitter sees the receiver, indicating a line of sight (LOS). Moreover, here we assume the transmitter sees the receiver, indicating a line of sight (LOS) channel, hence the signal leaves the transmitter towards to the receiver. The distance is d and here we do not assume any type of reflection. Figure (2.10) illustrates signal strength versus

distance for the path loss. The curve indicates the power loss of the signal due to traveling, which is attenuation in power. For this LOS channel (free space), the ratio of the power received P_r to the power transmitted P_t is given by [88]:

$$P_r = P_t \left[\frac{\sqrt{G_l}\lambda}{4\pi d} \right]^2 \tag{2.1}$$

where:

P_r : received power.

P_t : transmitted power.

G_l : the product of the transmit and receive antenna field radiation patterns.

λ: the wavelength.

d : the distance between transmitter and receiver.

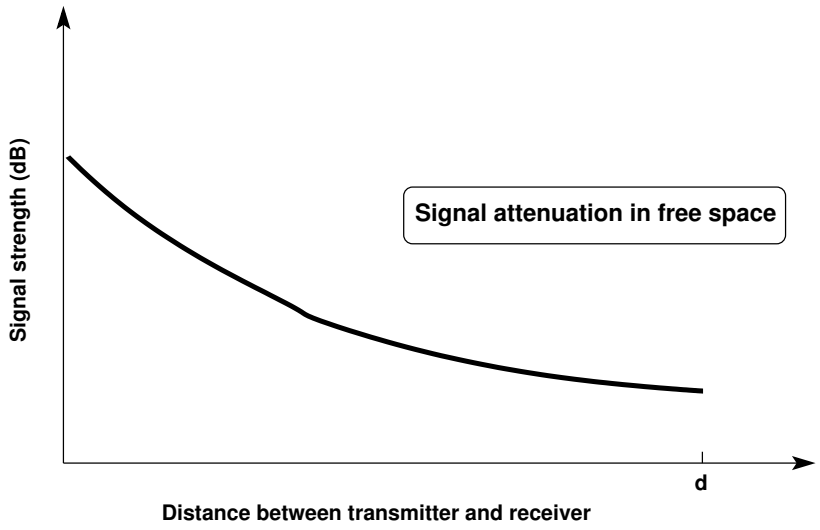


Figure 2.10.: Path loss versus distance

As indicated in Equation (2.1), the signal power decays inversely with the square of the distance from the transmitter. But in an indoor environment, this factor is increased, because of the presence of objects such as furniture and also because of destructive interference of the transmitted signal caused by the reflected signals from these objects. These combine to produce what is known as the path loss of the radio channel [89, 90].

On the other hand, only path loss can be treated as a deterministic effect, because it mainly depends on the distance between the transmitter and the receiver. It plays an important

role in larger time scales, such as seconds or minutes, because the distance between the transmitter and the receiver in most situations does not change significantly on smaller time scales [91]. Theoretically, as mentioned above, the signal power during travel from transmitter to receiver decreases in proportion to the square of the distance, as stated in Equation (2.1). But in practice, the power decreases more quickly, typically to the 3^{rd}or 4^{th} power of distance, because due to the earth's surface, some of the waves are reflected and may reach the transmitter with a phase shift of 180° and may then reduce the net received power. A simple two-ray approximation for path loss is shown in Equation (2.2) [91]:

$$P_r = P_t \frac{G_t G_r h_t^2 h_r^2}{d^4} \tag{2.2}$$

where:

G_r : Receiver antenna gains.

G_t : Transmitter antenna gains.

h_r : Antenna heights of receiver.

h_t : Antenna heights of transmitter.

In general, a common empirical formula for path loss is [91]:

$$P_r = P_t P_0 (\frac{d_0}{d})^{\alpha} \tag{2.3}$$

where:

P_0 : power at a distance d_0

α : path loss exponent.

The path loss is given by [91]:

$$PL(d)dB = \overline{PL}(d_0) + 10\alpha log(\frac{d}{d_0}) \tag{2.4}$$

where:

$\overline{PL}(d_0)$: the mean path loss in dB at distance d_0

b- Shadowing

Large scale channel variation on a mean level is known as shadowing. As the signal travels from the transmitter to the receiver it usually encounters large objects (obstacles) such as buildings, trees, cars and other objects. It will also drop in power; this is called log-normal fading. Hence, as indicated above, we assume free space and a medium with no obstacles or obstructions, but here the signal loses power because of obstructions [90, 91, 92].

For example, when a receiver is moving, an obstacle such as a large building may get between the transmitter and the receiver, causing an increase of the signal attenuation of the received signal. Some part of the transmitted signal is also lost through absorption, reflection, scattering and diffraction [90, 92] The receiver continues to move and, after a short time, the signal path is clear and the power of the received signal increases again. This effect is called shadowing and is illustrated in Figure (2.11) [91]. In this figure, if we imagine the transmitter antenna as a light source, the middle building would cast a shadow on the receiver antenna, hence the term shadowing. The net path loss becomes [87, 93]:

$$PL(d)dB = \overline{PL}(d_0) + 10\alpha log(\frac{d}{d_0}) + \chi \tag{2.5}$$

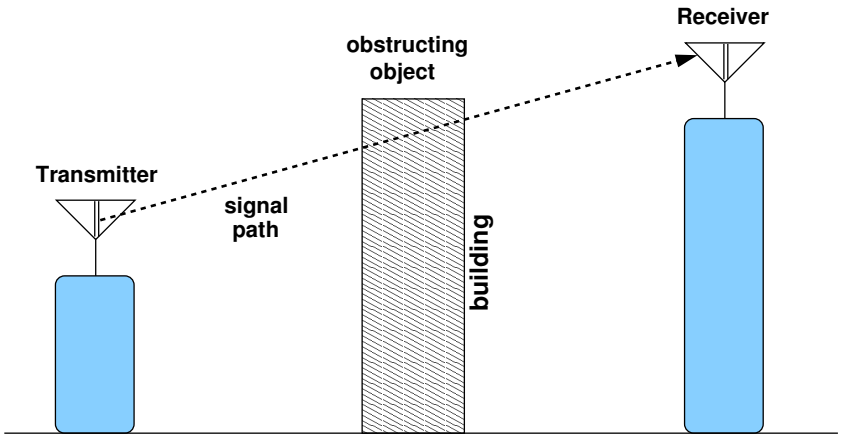


Figure 2.11.: Shadowing

Here χ is a normally (Gaussian) distributed random variable (in dB) with standard deviation σ, while the other variables are stated in Equation (2.4). χ represents the effect of shadowing. As a result of shadowing, the power received at the points that are the same distance d from the transmitter may be different and have a log-normal distribution. This phenomenon is referred to as log-normal shadowing; as mentioned above, there is not only one obstruction but probably several. We do not know how many; therefore, we must treat this problem as statistical or probabilistic rather than deterministic. This is why we see the

probability formula in the previous equation. Figure (2.12) shows the signal fluctuating depending on the obstacles, obstructions or objects it travels through. There is a decrease in power due to the long-normal fading [94].

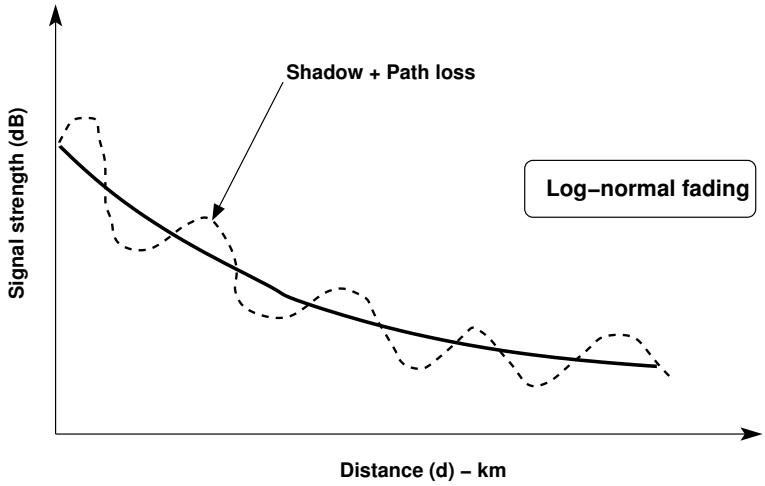


Figure 2.12.: The power drop due to slow fading (long-term fading)

c- Multipath Fading

Multipath fading is a phenomenon occurring in radio communications systems; at any point in time, the receiver receives multiple signals originating from the same transmitter, each of which following a different path between the transmitter and receiver, as shown in Figure (2.13) [71, 93, 95, 96]. Depending on the environment of the transmitter and receiver, there can be many or few objects reflecting the transmitted radio signal. In general these objects are known as scatterers, and the transmission of a signal leads to a situation known as multipath dispersion, or delay spread [73, 89, 97].

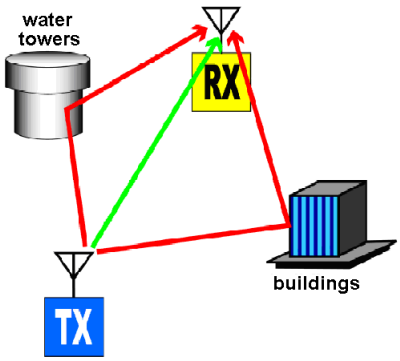


Figure 2.13.: Multipath fading scenario

- **Delay Spread:** This is basically the time between the first arrival and the last arrival of the signal. As a signal moves through the channel, it is reflected, diffracted and

scattered [77]. Here, reflected means each reflection represents a copy of the signal and arrives at a different time. The time between the first and last arrival is referred as a delay spread [81, 94, 98].

- **Doppler Shift:** This depends on how fast the transmitter is moving and how fast any object between transmitter and receiver, which would be responsible for signal reflection, is moving. The signal will encounter a change in frequency; this is called a Doppler shift [77]. If everything is stationary, the signal frequency characteristics will be preserved. Therefore, having a channel with a large Doppler shift means having a dynamic channel, where the transmitter and/or receiver and/or objects are responsible for scattering [95, 98].

2.2.3. Some Channel Models

The use of channel models for communication system design and evaluation is widespread and universally accepted as an important element of system optimisation [99]. In communication literature, a number of different methods have been proposed and used for the simulation of Rayleigh fading; some other channel models are based on noise and interference. In this section we briefly illustrate some of these channel models.

A- AWGN Channel Model

The Additive White Gaussian Noise (AWGN) communication channel is the predominant model used in the design and test of a communication system [100, 101]. In this model, the transmitted signal is corrupted by the AWGN signal. Usually, White Gaussian Noise (WGN) is needed for Digital Signal Processing (DSP) system testing or DSP system identification [77]. In [100], the author presents a novel scheme of implementing AWGN generators in FPGAs for channel emulators to evaluate the performance of communication systems. Compared with existing methods, like the Central Limit Theorem (CLT) method and the Box-Muller method, the proposed scheme has advantages in both speed and simplicity, especially when two independent emulators are needed, such as for modulation [87].

B- Rayleigh Channel

Rayleigh fading is a statistical model for the effect of a propagation environment on a radio signal. Rayleigh fading models assume that the magnitude of a signal that has passed

through a communications channel model will vary randomly, or fade, according to a Rayleigh distribution [89, 94, 97]. Rayleigh fading occurs because sometimes the transmitter moves, for instance when one uses a phone in a car or a train. In cases where the receiver moves or something between the transmitter and the receiver moves, the reflection can be from a moving target [101]. This gives rise to relative phase shifts between the received reflected signals, which can cause various reflected signals to attenuate the original transmitted signal (direct path signal), and may cancel each other out. This is called Rayleigh fading and is shown in Figure (2.14) [89, 94].

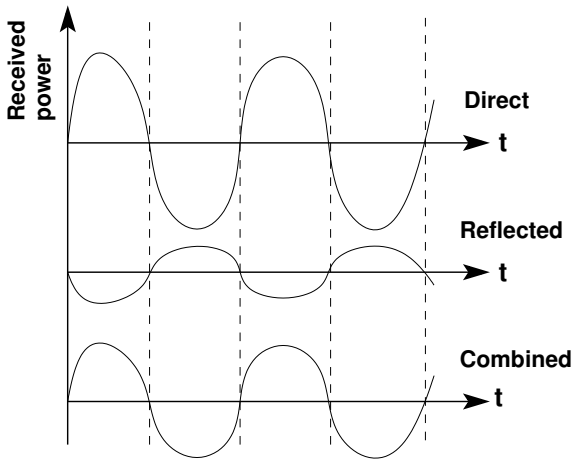


Figure 2.14.: Rayleigh fading

On the hand, Rayleigh fading is known as a signal impairment source. As shown in Figure (2.15), when we compare the path loss and shadowing effects mentioned above, we can see rapid variant fluctuation. Fluctuation of signal power is really a type of fading; therefore, we can see in the shadowing effect that the distance is 1-2km, while the distance here is a few meters; as discussed earlier, the signal fluctuation is very fast because the transmitter or the receiver is moving, or the reflectors responsible for reflecting the signal are moving [96].

The Rayleigh fading channel model belongs to a class of channels where the received signals of faded signals are based on Rayleigh distribution. This is mainly used for describing the statistical time varying nature of the received signal in an isotropic scattering environments, where no LoS propagation path exists between the transmitter and the receiver. The Rayleigh PDF is given by [102]:

$$P_{\alpha(t)}(y) = \frac{2y}{\Omega} exp(-\frac{y^2}{\Omega}), y \geq 0 \tag{2.6}$$

where $\Omega = E[\alpha^2(t)]$.

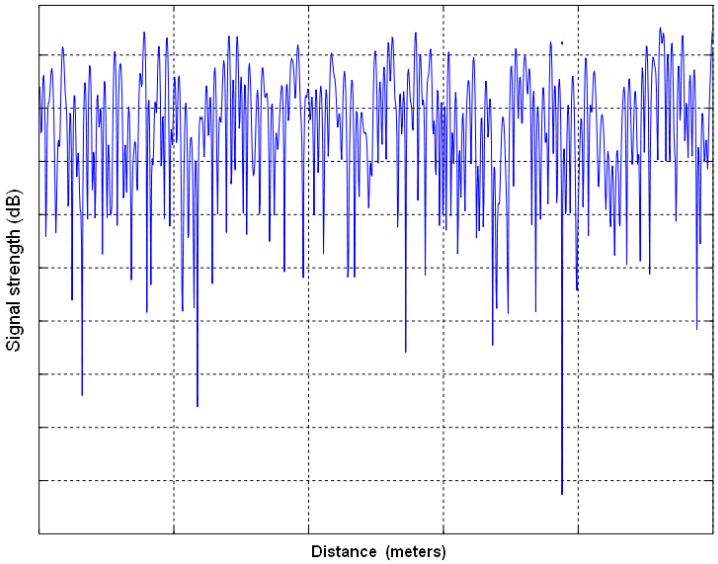


Figure 2.15.: Signal fluctuation in Rayleigh fading

α is a random variable.

C- Rician Channel

Rician distribution is commonly used to describe the statistical time varying nature of the received envelope, when a signal is transmitted over an environment where, in addition to many reflecting objects around the receiver, a LOS propagation route exists between the transmitter and the receiver [101]. It can also be used to describe the envelope distribution of the received signal when it contains a dominant non-faded component, although this dominant component is not the LOS one [96]. The Rician PDF is given by

$$P_{\alpha(t)}(y) = \frac{2(K+1)y}{\Omega} exp\left[-K - \frac{(K+1)y^2}{\Omega}\right] I_0\left(2y\sqrt{\frac{K(K+1)}{\Omega}}\right) \tag{2.7}$$

where : K represents the ratio of the power in the specular component.

$\Omega = E[\alpha^2(t)]$.

α is a random variable.

$I_o(.)$ is the 0^{th} order modified Bessel function.

2.2.4. Noise

The performance of any communication system is affected by noise and interference from other sources. In practice, noise represents the number of errors occurring in the system. It is the undesired electrical signals that always exist in communication systems [88]. It also can be defined as the spurious signals inserted into the communication signal by the channel, equipment, electromagnetic coupling or clicking of switches [99]. The performance of electrical systems is generally affected by noise. For this reason, in electronic communication systems the subject of noise and noise reduction plays a key role in system design and is also the most important single consideration in transmission of data [72]. But we can reduce the effect of noise through use of suitable filtering, the choice of a suitable modulation technique and the selection of an optimum receiver site. There are several types of noise that affect system performance, including thermal noise, impulse noise, crosstalk and intermodulation noise [72, 99]:

a- Thermal Noise

This is a natural source of noise, also called Johnson noise [99]. It occurs in all transmission media and all communication equipment. This type of noise cannot be eliminated because it is generated by the thermal motion of electrons in all passive devices, such as resistors, wires and so on. Electrons causing thermal noise are the same electrons responsible for electrical conduction [72].

From the previous description, we can claim that thermal noise is a statistical quantity and can be represented by its probability distribution. Also, Gaussian distribution is used to inform us of statistical randomness. Hence, thermal noise has a Gaussian distribution and exists at all frequencies, so it also called white noise, because it refers to the average uniform spectral distribution of noise energy with respect to frequency. The property of Gaussian distribution allows it to be completely specified in terms of its power density $(= power/Hz)$ and is usually denoted by $N_o = kT$, where k is the Boltzmann's constant and T is the temperature in ^oK (degrees Kelvin) [72, 99, 88].

b- Impulse Noise

This is another type of noise that plays an important role in the communications field. Impulse noise is different from other noise types because it is non-continuous (Figure (2.16)) and is formed from irregular pulses or noise spikes of short duration with relatively high amplitude [72, 99]. Sometimes these noise spikes are known as hits, and each spike

has a wide spectral content, which means the impulse noise can smear a large frequency bandwidth. Also, this type of noise may seriously degrade error performance on data or other digital circuits. This is due to the clicking of mechanical switches and results in either audible clicking sounds during conversation or spikes in digital bit transmission. Another source of impulse noise could be sparking, due to imperfect insulation [72, 88]. In this research, we consider impulse noise to model digital noise and then insert it into the system. The idea is to modify the individual bits or packets with a given probability, as we will see in Chapter Four.

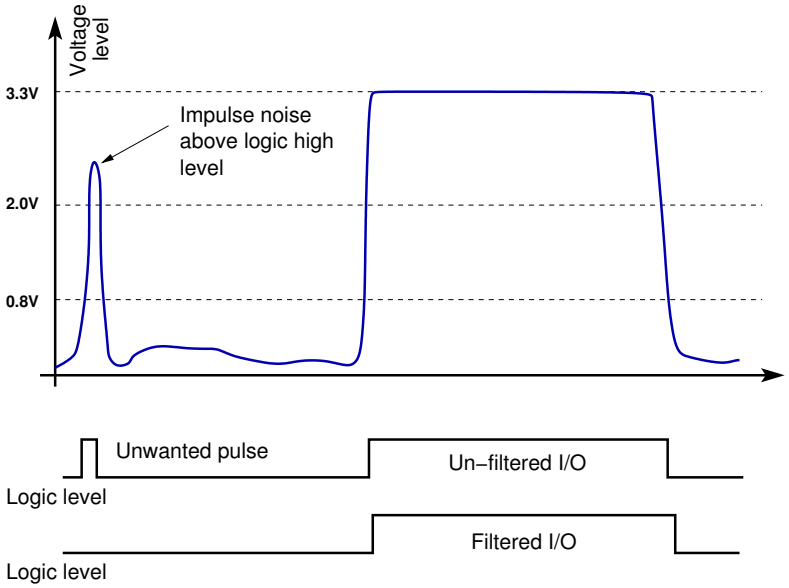


Figure 2.16.: Example of a digital signal suffering impulse noise and its corresponding logic level[2]

c- Crosstalk

This is a type of noise known as a disturbance, caused by electromagnetic interference producing an undesirable coupling over a cable pair. Therefore, the signals are confused and cross over each other. There are three main causes of crosstalk: electrical coupling between transmission media, poor control of frequency response and nonlinear performance in analog FDM [72, 99].

d- Intermodulation Noise

Intermodulation (IM) noise is a special type of crosstalk; it occurs as a result of the presence of intermodulation products. If we have two signals with frequencies F_1 and F_2 coming from two different circuits and moving through a nonlinear device, the result is in-

termodulate products which form a new signal that falls inside a frequency band and/or outside a frequency band. This type of noise is similar to harmonics in music [72].

2.2.5. Channel Model

In wireless networks, it is an important consideration to determine to what extent can a pair of nodes communicate. A link quality analysis has been carried out to describe the behaviour of the communication medium between the transmitter and the receiver terminals. This is based on the free-space propagation representation of the communication link in equation (2.8), which represents signal decay as a function of distance [88].

$$L_{fs} = 10Log_{10}\left(\frac{P_{tx}}{P_{rx}}\right) = -10Log_{10}\left(\frac{G_t G_r \lambda^n}{(4\pi)^n d^n}\right), \tag{2.8}$$

where the free-space propagation loss is L_{fs} (line-of-sight, in dB), P_{tx} and P_{rx} are the transmitted and received power in Watts, G_t and G_r represent the respective antenna gains and the link distance is d. The receiver sensitivity required is usually quoted in dBm, and can be computed using equation (2.9) [103].

$$P_{rx} = P_{tx} - L_{fs} - Fade\ Margine \tag{2.9}$$

The general expression for propagation loss in dB with the assumption that the antenna gains are 0 dB (for simple dipole antennaes) and in free space (where n is assumed to have the value of 2) can be expressed as [88]:

$$L_{fs} = P_{rx}[dB] - P_{tx}[dB] = 10 \times n \times Log_{10}(^{4\pi^d}/_{\lambda})\ dB \tag{2.10}$$

where $\lambda = {}^{c}/_{f}$ is the free-space wavelength at the carrier frequency (and c is the speed of light and f is the frequency).

Signal reduction due to multipath fading is normally in the range of 20 to 30 dB and hence in practice a fade margin will be added to the power loss to account for it [88, 103].

We proceed to incorporate the effects of the modulation technique employed. This will allow us to estimate the biterror probability P_B which represents the Bit Error Rate (BER) as a function of the bit energy and noise-density of the signal ${}^{E_b}/_{N_0}$. Figure (2.17) illustrates the waterfall like shape of most such curves. For the purpose of link budget analysis, the

most important aspect of a given modulation technique is the signal to noise ratio necessary for the receiver to achieve a specified level of reliability in terms of P_B (BER) .

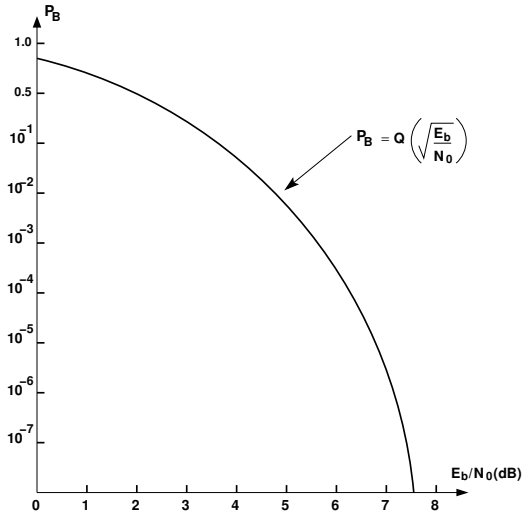


Figure 2.17.: General shape of PB versus E_b/N_0curve

The probability of a bit error, P_B, is defined as [88]:

$$P_B = Q(z) = Q\left(\sqrt{\frac{E_b}{N_o}}\right) \tag{2.11}$$

Where $Q(z)$, is called the complementary error function or co-error function. This complementary error function is numerically equal to the area under the "tail of the Gaussian". It is closely related to the complementary error function $erfc(z)$ and error function $erf(z)$:

$$erf(z) \equiv \frac{2}{\sqrt{\pi}} \int_0^z e^{-x^2} dx, \qquad z \geq 0 \tag{2.12}$$

$$erfc(z) \equiv \frac{2}{\sqrt{\pi}} \int_z^\infty e^{-x^2} dx = 1 - erf(z), \qquad z \geq 0 \tag{2.13}$$

The $Q - function$ is related to these functions by

$$Q(z) = \frac{1}{2} erfc\left(\frac{z}{\sqrt{2}}\right), \qquad z \geq 0 \tag{2.14}$$

Equation (2.15) provides a sample BER model for a specific modulation scheme, Quadratic Phase-Shift Keying $(QPSK)$. The BER assuming white noise AWGN is given by [103, 71]:

$$BER = \frac{1}{2} erfc\left(\sqrt{E_b/N_0}\right) \tag{2.15}$$

where the maximum thermal noise power within a given bandwidth B is given by $N_0 = kTB$.

In the radio and microwave bands, the spectral density is taken as N, for a one sided spectrum, and as $\frac{N_0}{2}$ for a two side spectrum, where: k = Boltzmann's's constant, T = system temperature in Kelvins, usually assumed to be $290K$. SNR gives the relation between the received signal power S and the noise power N as given by [88]:

$$SNR[dB] = 10 \times Log_{10}\left(\frac{S}{N}\right) \tag{2.16}$$

Where E_b/N_0, in equation (2.15) is just a normalised version of SNR [88], which can be rewritten to emphasise that E_b/N_0, is just a version of S/N, normalised by bandwidth and bit rate R, as follows [88]:

$$\frac{E_b}{N_0} = \frac{S}{N}\left(\frac{B}{R}\right) \tag{2.17}$$

2.3. Flocking Behaviour System

As previously mentioned, one phase of this research is to create a demonstration in order to validate the wireless methodology by developing a small application and/or test case. A flocking behaviour system is selected as a case study for demonstration, because we must prove that incorporating and fixing the wireless channel, wireless protocol, noise or all of these factors early in the developed SystemC design methodology (wireless methodology) is very advantageous, i.e. to show that small changes in the wireless specifications result in big changes in system dynamics. Therefore, we may need to construct different architecture to investigate the system over different performance parameters. Our target is not just designing and implementing a flocking behaviour system: the implementation of such a system is trivial and has already been done a number of times [104, 105, 106]. The goal of this case study is to show that communication can have a big impact on system dynamics. This is achieved by incorporating communication early in the design space to create an optimal design. Moreover, we need to optimise system stability in terms of communication.

2.3.1. What is Flocking?

A flock is a phenomenon defined as a large group of animals organizing and moving together into an ordered motion. This group can be a enormous flock of birds moving with a collective and coherent motion, a number of wild animals moving across the savanna, a school of fish migrating, swarming of bacteria, and so on [104, 107, 108]. The main characteristics of flocking systems are: distributed control, local interaction and self-organisation [109]. The reason for creating a flock is that its members are believed to have certain advantages over individuals, such as increased safety from predators [110, 111] and better chances of finding food [109]. Though people have always been intrigued by the movements presented by different flock types, modelling and systematic studies of flocking behaviour have only recently emerged. The first real effort to simulate and model flocking behaviour was done by computer scientist and animation artist Craig Reynolds in 1987 [112]. However, since that time flocking has become a subject of great interest in many fields, such as computer science, biology and applied mathematics [106, 113].

2.3.2. Flocking Applications

Understanding flocking behaviour concepts can help designers construct and implement many artificial autonomous application in different fields, such as unmanned air vehicles (UAV), autonomous underwater vehicles (AUVs), search systems, mobile robots, etc.[109, 114].

2.3.3. Reynolds's Model

In 1987 computer scientist Craig Reynolds presented a flocking behaviour model to simulate the motion of a flock of birds [112]. He simulated flocks by starting to model each individual member of the flock, known as a boid. A boid is a flock member and can be any entity which can participate in the flocking behaviour, such as a bird, fish or sheep [107, 108]. In Reynolds's model, each boid in the flock makes steering decisions based on three rules [112]. These are:

- **Separation:** This refers to collision avoidance. Each boid always tries to steer away from other boids near them.
- **Cohesion:** This is when boids move toward the average position of local flockmates.
- **Alignment:** This refers to velocity matching. The boids must match their velocity to that of other boids.

With these fundamental rules, a simple flocking behaviour system can be implemented as described in [104, 105, 106]. Accordingly, in [104], Travers develops a simple algorithm to simulate a flocking behaviour by using autonomous agents with simple movement rules. He designs his algorithm based on the three rules proposed by Craig Reynolds [112]. Motivations for studying flocking phenomenon in the biological world are described in [106]. The work in [105] constructs a model that can be used for real-time simulations and also has the potential to be used in games, simulations, crowd descriptions, etc. In 1999 [115], Craig Reynolds advanced his original model, implemented in 1987. This extension is a greatly expanded model that is easier to fit and implement. In recent years, problems in the flocking phenomenon have attracted much attention among researchers working in the fields of computer science, control engineering, biology and physics [116, 117, 118]. As a result, flocking has opened a new subject on how to investigate the cooperative movements of a large group of agents without a centralized scheme, and how to develop various cooperative control capabilities of engineering groups, such as control of mobile robots, design of mobile sensor networks and unmanned aerial vehicles (UAV) [119].

2.3.4. Development of Flocking Behaviour Models

In recent years, researchers have suggested some changes to improve the rules described above, including obstacle avoidance and goal seeking. Many algorithms have been proposed to achieve these rules. In [114], Zonggang uses graph theory as a basic tool to investigate and solve the coordinated control problem, because this theory provides an approach to explaining the relations between the coordinated variables. Also, the leader/follower approach to the goal tracking problem is a research topic investigated in [111, 120, 121, 122, 123, 124].

2.4. Summary

In this chapter we gave background information about the main areas that required investigation in order to develop a SystemC design methodology to model and simulate wireless communication systems. In the first part, we described the present SystemC design methodology, noting its strengths in describing hardware in a unified fashion from the level of untimed functional models down to RTL level models, and the benefits that gives designers in modelling at differing levels of abstraction. We reviewed recent examples in the literature of SystemC being used to design large-scale heterogeneous systems and networks, and modern attempts to extend SystemC to the modelling of analogue and mixed signal systems (including one example of the use of SystemC in RF transceiver design).

However our review did not find any comprehensive examples of SystemC being used in the comprehensive design of wireless networks, a gap in the literature that this research aims to fill.

The second part first introduced the theory of radio communication systems and then provided background on channel modelling methods, by presenting some channel models such as AWGN, Rayleigh and Rician. This information was necessary in order to understand how a wireless communication channel model, which forms the core of any wireless communication system, can be modelled and developed in order to use it as a building block for the development of a SystemC methodology to support wireless communication. The results of this discussion clearly showed the need for noise and delay to be included in the channel model in order to investigate system performance under real conditions.

Finally, we explained the concepts behind the flocking behaviour system selected as a case study to demonstrate our methodology. It is a very complex system and we need to model it based on the developed methodology in order to prove that incorporating and fixing the wireless channel, wireless protocol, noise or all of these elements early in the design methodology is very advantages, because our target is not just designing and implementing a flocking behaviour system. The implementation of such a system is trivial and has already been done a number of times. The goal of this case study is to show that communication can have a big impact on system dynamics as we will show in the coming chapters.

Part II.

SystemC Wireless Methodology

3. RTL-Level Modeling of an 8B/10B using Existing SystemC Methodology

As mentioned before, we need to construct some standard components at the system level that are typically required to implement wireless communication systems; one of these standard components is an 8B/10B Encoder/Decoder. Therefore, this chapter presents an RTL-level model of an 8B/10B encoder/decoder block in SystemC. The use of 8B/10B coding is an important technique in the construction of high performance serial interfaces. These are particularly suitable for alleviating the I/O bottleneck of state-of–the-art systems. SystemC has been chosen because it provides a homogeneous design flow for complex designs (i.e. SoC and IP based design), where system modelling at the early stages of the design becomes increasingly important.

3.1. Introduction

Serial transmission technology is increasingly used for the transmission of digital data. State of the art communication networks make use of serial links for transferring data [22]. This is in part due to the reduction in pinout and cost; but most importantly because it is inherently immune to skew, which plagues high speed parallel interfaces. To improve the performance in serial data transmission systems, block coding is used to ensure sufficient data transitions occur for clock recovery and also to help guard against errors. In the early 80's the 8B/10B block coding technique was introduced by Albert Widmer and Peter Fransazek of IBM Corporation [125]. Although quite old, the technique continues to be employed in state of the art technologies, such as HyperTransport, IEEE1394b, SATA, DVB, and many others.

This chapter describes the construction of an RTL model of an 8B/10B encoder in SystemC. Although other HDL models have been used, a SystemC model is desirable as it

integrates into the SystemC design methodology, which provides a consistent framework for the design and modeling of complex systems at numerous levels of abstraction (which encompass all levels from system specification to implementation). This is particularly important for the design of SoC systems, where it is necessary to determine the system performance before a prototype is constructed, in order to evaluate the merit of different implementations. Also, because SystemC can describes hardware at high levels of abstraction, it also provides a faster simulation time, when compared to VHDL [52], which is an important feature when a large design space needs to be explored. Managing abstraction, early verification and using the homogeneous model are the strongest weapons in combating complexity [8].

The majority of published work is centered towards specific 8B/10B implementations, such as in [126] which perform 8B/10B encoding/decoding within Lattice programmable logic devices (PLDs). In [127], Wu et al propose a new peak-to-average power ratio reduction method. They use an 8B/10B code in the time domain of OFDM system to reduce peak-to-average power ratio. DC-balance is ensured because the encoder transmits the same number of ones as zeros. This is important, as providing AC-coupled links eliminate the ground-loop problem. In this novel work, the main objective is to develop a reusable 8B/10B IP core offering flexible interoperability, which can be used to allow efficiently prototyping of serial communication systems. Moreover, the model has been constructed at the RTL level so that it can be efficiently synthesized to hardware, or implemented as a software component if required [44, 128].

The remainder of this chapter is organized as follows: Section two provides a brief description of the 8B/10B Encoder-Decoder. Section three describes the SystemC 8B/10B Encoder-Decoder Model. Software implementation is discussed in sections four, five and six. Section seven provides results and a brief analysis. Finally, conclusions are drawn in section eight.

3.2. 8B/10B Encoder-Decoder Description

In an 8B/10B encoding process a block of 8 bits of data is converted to a 10-bit block before transmission (Figure(3.1)), with the additional information used to ensure that i) enough data transitions are present. ii) DC balance is achieved and iii) to aid in providing data integrity. The decoder decodes a 10-bit code into 8 bits of data. For ease of reference, the eight input bits are named A, B, C, D, E, F, G, H, where A is the least significant bit (LSB), and bit H is the most significant bit (MSB). They are split into two groups in the encoding process: The five-bit group A,B,C,D,E, and the three-bit group F,G,H. The

coded bits are named a, b, c, d, e, i, f, g, h, j (the order is not alphabetical). These bits are also split into two groups in the decoding process: the six-bit group a,b,c,d,e,i, and the four-bit group f,g,h, j.

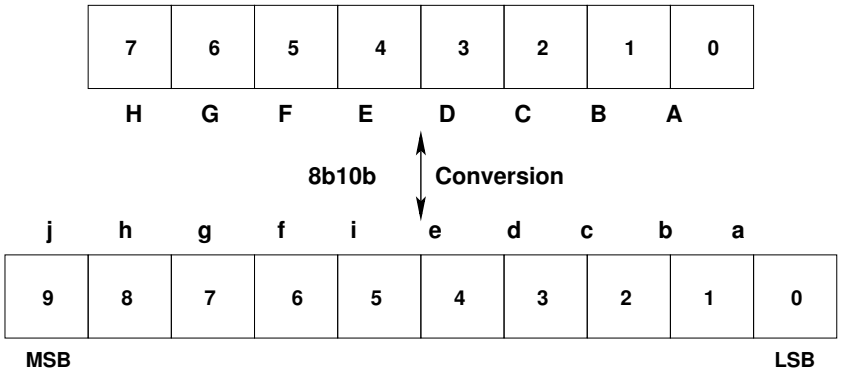


Figure 3.1.: 8B-10B Conversion

Figure(3.2) shows a diagram of an 8B/10B Encoder/Decoder block. Since 8 bits of data are converted to 10 bits before transmission, the technique requires transmitting encoded data 25% faster than the desired throughput (i.e. 1 Gb/s of encoded data is transmitted at 1.25 Gb/s between terminals [129]). Structurally the 8B/10B code is defined from simpler 5B/6B and 3B/4B codes. For instance, in the encoding side, the 8B/10B Encoder consists of two sub-blocks, the 5B/6B and the 3B/4B encoders, shown in tables (3.1) and (3.2), respectively. In order to aid in clock recovery, the code is designed so that no more than five consecutive 0's or 1's are ever transmitted [125]. The 8B/10B encoder block continuously converts the incoming data to 10-bit symbols. The conversion is done depending upon the value of a signal called *running disparity* and the incoming stream of data. The running disparity is a binary parameter, which has either a positive or a negative value, and whose purpose is to ensure DC balance is maintained in the stream. The running disparity of any incoming data is calculated based on the number of logic 1's and 0's present in that data code group. On reset the running disparity value is initialized as negative.

An 8B/10B encoder takes a one byte input, and generates a 10-bit code. Some of these codes are balanced (i.e. they have an equal number of 1s and 0s), while others have a disparity of ± 2 (either four 1s and six 0s, or, six 1s and four 0s). These last codes are always assigned in pairs, such there are always two symbols (with disparities of +2 and -2 respectively) associated to that particular input. The disparity (if any) of the current, and any previous symbols is tracked by the running disparity variable, whose purpose is to maintain an overall balanced stream. This is achieved by selecting the proper encoding symbol so that the running disparity is held at ± 1 (the running disparity is initialized at reset to -1 [125]). The decoder, on the other hand, converts 10-bit symbols to 8-bit

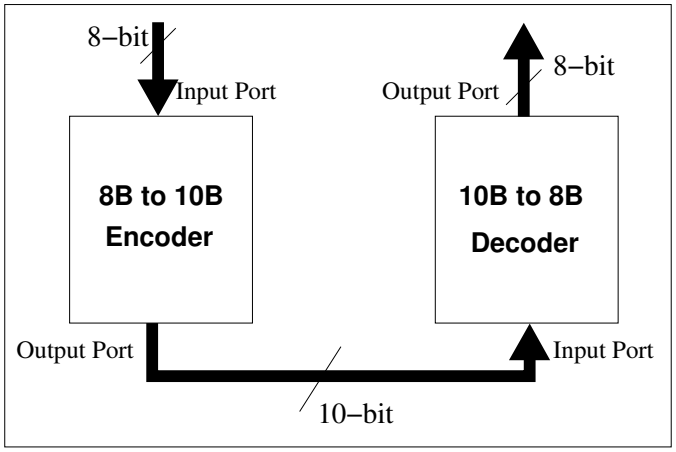


Figure 3.2.: 8B/10B Encoder/Decoder Block Diagram

data, but does not need to track the running disparity, other than for synchronisation and error correction purposes, as in this case the mapping is surjective. For completeness the decoder implementation described in this chapter does keep track of the disparity.

3.3. 8B/10B Encoder-Decoder SystemC Modules Structure

SystemC defines a system modeling and design methodology, which is supported by a C++ class library, that can be used to model systems in a homogeneous environment all the way from requirement capture to system partitioning, cycle accurate modeling and backend implementation. This allows to obtain performance metrics at high levels of abstraction which can be used to asses the impact of different architectural solutions early in the design phase [30, 8].

In this work, structural designs for the encoder and decoder models are implemented in SystemC using modules, ports, processes and signals which represent the fundamental constructs of SystemC libraries. The Modules can contain other modules, allowing a hierarchical construction of the system model. Processes communicate to each other via interfaces, channels and ports, and can synchronize with each other via event objects. Also there are a variety of data types are supported to include single bits, bit vectors and fixed-point integers[13, 130]. The encoder and decoder models have been implemented by using modules and can be connected together through the ports which are created from the base class sc_in<data_type>in_port_name and sc_out<data_type> out_port_name.

Decimal	Binary	Codeword
0	00000	100111 or 011000
1	00001	011101 or 100010
2	00010	101101 or 010010
3	00011	110001
4	00100	110101 or 001010
5	00101	101001
6	00110	011001
7	00111	111000 or 000111
8	01000	111001 or 000110
9	01001	100101
10	01010	010101
11	01011	110100
12	01100	001101
13	01101	101100
14	01110	011100
15	01111	010111 or 101000
16	10000	011011 or 100100
17	10001	100011
18	10010	010011
19	10011	110010
20	10100	001011
21	10101	101010
22	10110	011010
23	10111	111010 or 000101
24	11000	110011 or 001100
25	11001	100110
26	11010	010110
27	11011	110110 or 001001
28	11100	001110
29	11101	101110 or 010011
30	11110	011110 or 100001
31	11111	101011 or 010100

Table 3.1.: 5B/6B Encoding

Decimal	Binary	Codeword
0	000	0100 or 1011
1	001	1001
2	010	0101
3	011	0011 or 1100
4	100	0010 or 1101
5	101	1010
6	110	0110
7	111	0001 or 1110

Table 3.2.: 3B/4B Encoding

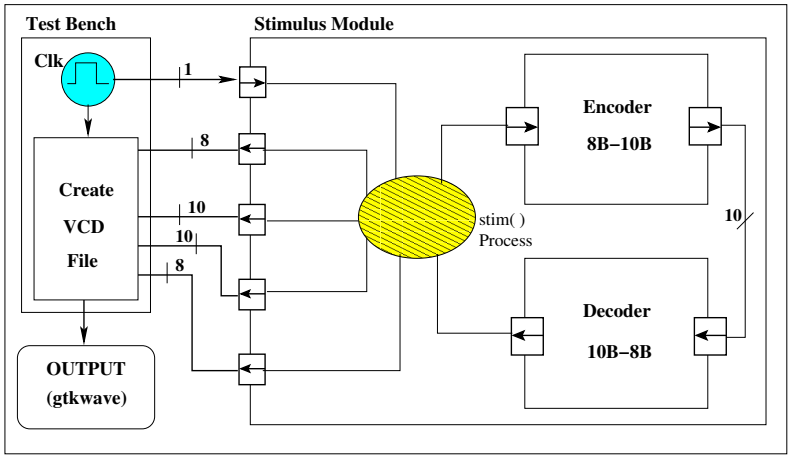


Figure 3.3.: Block Diagram illustrates Program Implementation Structure with all Designed Modules

3.4. Software Implementation

The overall block diagram of the system is depicted in Figure (3.3). The system is implemented using three main modules: The 8B/10B encoder and decoder modules, which are the devices under test (DUT), the testbench and the stimulus modules. As depicted, the stimulus module has been created as hierarchical construction which instantiates both the encoder and decoder modules. It generates a byte wide data stream of random 1s and 0s then send them to the encoder module input port. Then, the encoder module maps the 8-bit parallel data input to 10-bit output DC balanced stream of 1s and 0s. This 10-bit output is then loaded in and shifted out through its output port which is connected directly to the input port of the decoder module. Next, the decoder module collects the encoded data from its input port and then re-map the 10-bit data back to the original 8-bit data. There are five ports that are created in the stimulus module, one is the input and the other are outputs. All of these ports are used by the test bench which manages the designed modules, captures data for visualisation purposes and verifies the correct operation of the system.

To create the test bench, the main program links-in the various modules and interconnects them. It generates a clock source that need for simulation and apply it to the design and collect output waveforms. Through the four output ports of the stimulus module input data and output data packets (before and after encoding and decoding) of the encoder and decoder have been collected and sent to (Value Dump) VCD file which can be read by GTKWave wave viewer program to display output results.

3.4.1. The 8B/10B Encoder

The 8B/10B encoder is implemented as described in [125]. The first step is to apply the data to the input ports of the encoder module. The input data is represented by an eight bit data line, and a one bit control line, K, which indicates whether the input lines represent data or indicates that the character input should be encoded as one of the 12 allowable control, or K characters (refer to Table 3.3 for description of the core I/O pins). Each incoming byte is partitioned into two sub blocks and applied to a 5B/6B and a 3B/4B encoder respectively. A disparity control block (shown in Figure (3.4) and on the program segment shown in 3.1) controls the encoding. In this implementation, the decoder exhibits a five clock cycle latency. This information is readily available to the designer at the early stages of design, which is an advantage of using the SystemC methodology.

Name	Type	Description
CLOCK	IN	Clocks all the encoder logic. all input ports have their time referenced to the rising edge of the clock input.
RESET	IN	Global asynchronous reset (active high)
KI	IN	Control (K) input(active high) , is used to indicate that the input is for a special character
HI,GI, FI,EI, DI,CI, BI,AI	IN	Declare unencoded 8-bits Input Data, these 8-bits are named ABCDE_FGH as they represent a 5-bit and a 3-bit sub block of the encoder, where A is the LSB, and H is the MSB.
AO,BO, CO,DO, EO,IO, FO,GO, HO,JO	OUT	Declare Encoded 10-bits output Data, these 10-bits are named ABCDEI_FGHJ as they represent a 6-bit and 4-bit sub blocks, where they arranged from Least significant to Most.

Table 3.3.: Encoder Signals Definition

Algorithm 3.1 Program segment of 8b/10b Encoder

```
SC_MODULE (enc_eight_ten) {
sc_in<sc_logic> RESET;
sc_in<bool> SBYTECLK;
sc_in<sc_logic> KI;
sc_in<sc_logic> AI,BI,CI,DI,EI,FI,GI,HI;
sc_out<sc_logic> AO,BO,CO,DO,EO,IO,FO,GO,HO,JO;
sc_signal<sc_logic> XLRESET, LRESET;
sc_signal<sc_logic> L40 ,L04, L13, L31, L22;
...
...
sc_signal<sc_logic> NFO, NGO, NHO, NJO, SINT;
// 5b Input Function
void enc_5b(void);
...
...
void ENC3B4B (void);
SC_CTOR(enc_eight_ten) {
SC_THREAD(SYNCRST);
sensitive << XLRESET << SBYTECLK << RESET;
dont_initialize();
...
...
SC_THREAD(ENC3B4B);
sensitive << LRESET << SBYTECLK << COMPLS4;
dont_initialize();
}};
...
...
void enc_eight_ten::enc_5b(void) {
while(true) {
wait(1);// Wait events in sensitivity list
// Four 1's
L40.write(AI & BI & CI & DI); // 1,1,1,1
...
...
L22.write((~((sc_logic)AI) & ~((sc_logic)BI) & CI &
DI)|(~((sc_logic)AI) & BI & CI & ~((sc_logic)DI))|(AI & BI &
~((sc_logic)CI) & ~((sc_logic)DI))|(AI & ~((sc_logic)BI) &
~((sc_logic)CI) & DI)|(~((sc_logic)AI) & BI & ~((sc_logic)CI) & DI)
|(AI & ~((sc_logic)BI) & CI & ~((sc_logic)DI)));
wait(2);
}}
```

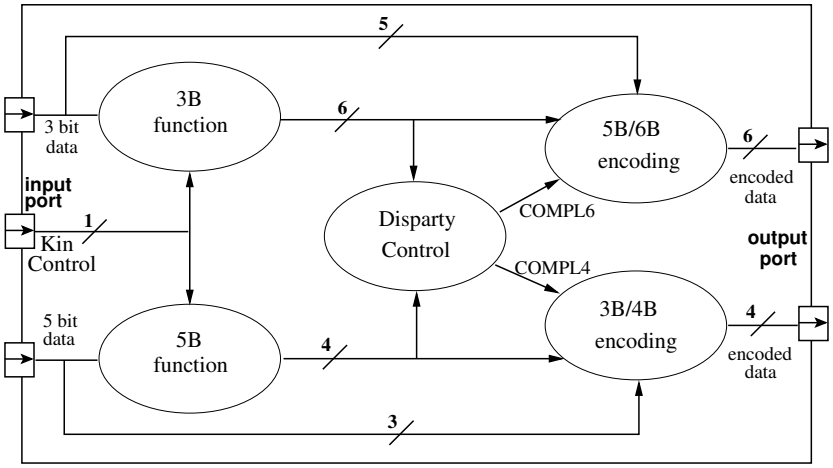


Figure 3.4.: 8B/10B Encoder Module

3.4.2. The 10B/8B Decoder

The 10B/8B decoder converts 10-bit symbols received from the encoder side to 8-bit data. The received code are decoded based on the running disparity process, as previously stated. The block diagram of the decoder module is shown in Figure (3.5). Latency for this modules is the same as for the encoder; i.e. 5 cycle latency, after which the decoded data is passed to the decoder output port and collected by stimulus process.

3.5. Results and Discussion

The system described in 3.3 was implemented in SystemC. The testbench was used to verify the correct operation of the system and to capture data to aid in visualisation as previously described. Figure (3.6) shows a simulation timing diagram for the 8B/10B encoder/decoder operating on random data generated by the stimulus block. For simulation

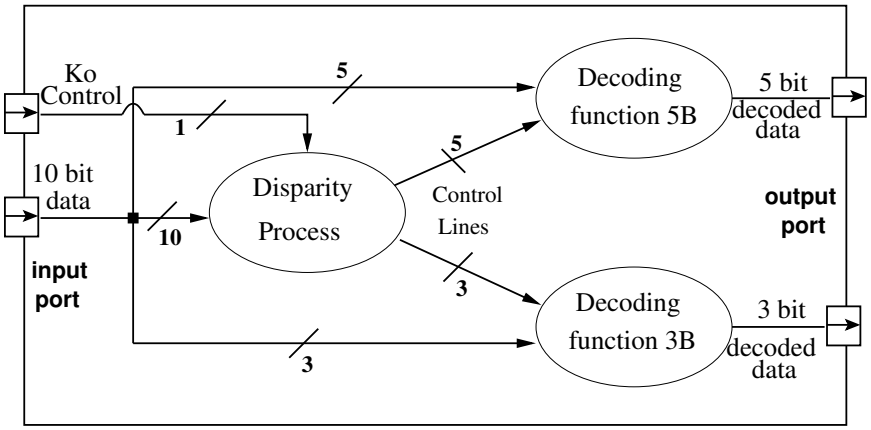
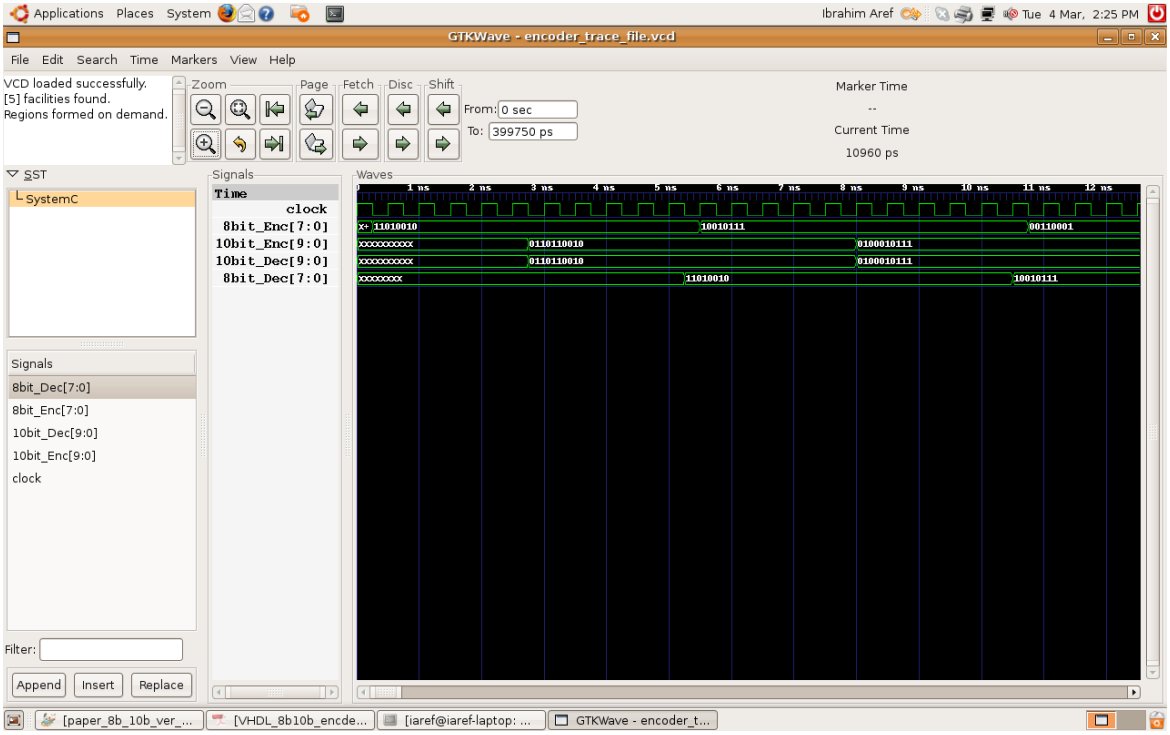


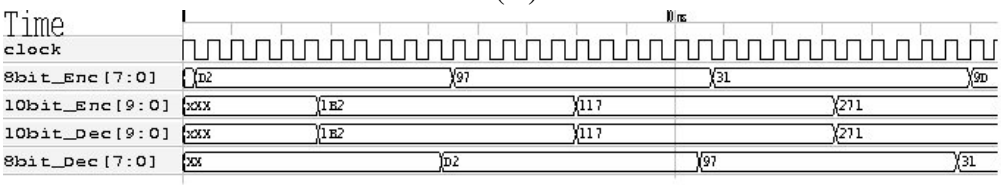
Figure 3.5.: 10B/8B Decoder Module

purposes the clock frequency is fixed at 200 MHz. As mentioned in section 3.4, there is a five clock latency for both, the encoding and decoding purposes, which is apparent in the simulation. Furthermore, as the implementation of the stimulus module has not been pipelined, throughput is limited by the ten cycle round-trip latency. This is an accurate representation of bursty communication systems with short packets.

All signals in the designs are latched on the rising edge of the clock.In a second test, the 8B/10B encoder/decoder successfully completed an exhaustive test, to cover all possible input vectors.



(A)



(B)

Figure 3.6.: Encoder Decoder Simulation Timing Diagram

3.6. Summary

In this chapter, an RTL-level SystemC model of an 8B/10B Encoder/Decoder core has been described. Although other HDL models of 8B/10B decoders have been published,

to the best of our knowledge, none target the SoC and IP design methodologies. SystemC has been chosen, as it provides a homogeneous platform for the design and modeling of complex systems. Furthermore, as systems become more tightly integrated (as in for example SoC) the ability to evaluate the system performance at early stages of a design becomes increasingly important. This is facilitated by the SystemC design methodology, and by following an IP-based design.

Although implementation and prototyping of the core is out of the scope of this work, it is important to note that both are relatively simple tasks, as the model developed is already at the RTL-level. The implementation can be carried out by exporting the SystemC code into VHDL or Verilog for synthesis, or directly by using the synthesisable subset of the SystemC language. Once the design has been mapped into a target technology, it is possible to back-annotate timing information directly into the SystemC model, which can aid in providing more accurate timing information for the high level modeling of complete systems.

4. Developing SystemC Design Methodology to Support Wireless Systems

The conventional SystemC design methodology is employed in order to manage the complexity of the design flow process at the system level [41]. For example, it was successfully employed to describe network-on-chip architecture [131] and to model the lowest network layers of the Bluetooth communication standard [132]. As mentioned previously, this work evaluates the potentiality of SystemC in simulating wireless communication systems in order to provide capabilities for co-simulating HW, SW and communication. We will show how the existing SystemC design methodology is developed and employed to model complex wireless communications systems in SystemC by modelling the communication nodes and connecting their instances to the wireless communication channel model [133] that reproduces the behaviour of wireless features such as propagation delay, interference, collisions and path loss. The design of the node can be dealt at different abstraction levels, from the system level down to the register transfer level. After each refinement step, nodes can be tested in their communication environment to verify that their communication constraints are met. Communication nodes with different functionality can be mixed in the simulation, thus allowing the exploration of communication scenarios made of complex devices. Synthesis can be directly performed on those models, provided they are described using a suitable subset of the SystemC syntax. This chapter demonstrates how to incorporate wireless features into conventional SystemC methodology.

4.1. Models Needed for SystemC Wireless Methodology

SystemC lacks elements that can be used to model and simulate wireless communication systems because it does not support noise links natively (wireless features). Therefore, a way to make SystemC support wireless systems must be established. Based on the

information described in Chapter One and Chapter Two, we need to address three issues in order to enable SystemC to support wireless communication systems:

1. Model a noisy wireless communication channel.
2. Construct other parts of wireless systems.
3. Create a demonstration in order to validate the developed methodology.

4.1.1. Modelling of a Noisy Wireless Communication Channel

The most important model needed to extend SystemC methodology to implement wireless systems is a wireless channel model. We have modelled a wireless communication channel in [133] that fulfils SystemC language requirements to support wireless systems; this is described in detail in Chapter Four. The main aim of modelling this channel is to change SystemC to include wireless features. The model supports the setting of a different Signal-to-Noise Ratio (SNR) and different types of interference. The channel specifications support wireless features such as noise. The simplest model of digital noise is simply to consider the constructed impulsive noise, in which individual bits or packets are modified with a given probability. Though other distributions can be used to model the noise, we use exponential distribution for the sake of simplicity. The noisy digital wireless channel is modelled successfully with a preset Bit-Error-Rate (BER) for Point-to-Point (P2P) and Point-to-Multipoint (P2M) platforms based on SystemC. This channel model represents what is missing in the SystemC language to support wireless systems.

4.1.2. Constructing Other Parts of Wireless Systems

We need to construct some standard components at the system level that are typically required to implement wireless communication systems. As mentioned in the first chapter, some of these standard components are PLL, 8B/10B Encoder/Decoder, MAC, communication protocol such as HDLC, and modulation techniques. An example of one of the wireless parts is an 8B/10B encoder decoder, which represents some form of encoding and decoding is selected to model up to RTL level; we examined it in detail in Chapter Three. Moreover the work on 8B/10B encoders/decoders was necessary to develop the skills necessary to develop the methodology of the thesis. As mentioned before, It is an 8-bit 10-bit Encoder/Decoder that is typically required to implement a wireless communication system. It is a standard model constructed to further understand how we use SystemC methodology to design and model a system. Also, it is an important technique in the

construction of high performance serial interfaces. In this novel work, the main objective is to construct an 8B/10B model at the RTL level, so that it can be efficiently synthesized to hardware, or implemented as a software component if required. Hence the model can be used to allow efficient prototyping of serial communication systems. Moreover, it has not been previously modelled at RTL level, so we have modelled it using SystemC in [134].

4.1.3. Creating a Demonstration

In the final stage, we need to validate and prove that our developed methodology can be used to model complex wireless communications systems, so it is applied to a case study consisting of the modelling of a flocking behaviour system; this is illustrated in part three.

As a first phase, we need to model and implement the first and second issues, because they represent the requirements to develop wireless methodology and they are not part of SystemC. This chapter focuses on developing SystemC design methodology to support wireless systems and examines how to integrate the three issues mentioned above to achieve the main goal of the research.

4.2. Wireless Extension into Existing SystemC Methodology

Complex wireless communication systems set various challenges in modelling and design fields. Inserting Communication as a new dimension in the design flow process causes conventional SystemC methodology tasks to become more complex. Modelling and simulating communication features and their interaction with System aspects in the early stages of design flow is thus key for developing an effective design methodology (which is our target in this research).

In previous works, such as [133, 134], the system alternative (HW and SW) and its many issues, such as partitioning options, architectural issues, synthesis and validation [29, 46], have been fully studied. The integration between System and Communication modelling in the SystemC platform is a new subject, because no one has extended SystemC to model wireless systems yet. On the other hand, the Communication aspect gives the system designer additional freedom in selecting how to implement a certain function. Given the different abstraction levels of the System and Communication alternatives, this leads to a

Naïve structure, which means one partitioning step or can be lead into hierarchical partitioning steps in the System/Communication aspects. In both cases, Communication becomes a new branch, as the System aspect is in traditional SystemC methodology.

4.2.1. System Design Flow Based on Naïve Structure

In this approach, which is called the Naive approach, the whole system design space is represented by HW/Node, SW/Node and Functionality/Node, as shown in Figure (4.1). For each HW/Node we have a Functionality/Node and for each SW/Node we have a Functionality/Node; our solution thus becomes very complicated, because the design space is very large. However, in the architecture exploration phase we are free to move anywhere we want in the whole system design space. In this case, the design flow is more flexible but the problem is that the design space becomes huge, as illustrated in Figure (4.1). We cannot use architecture exploration to find the optimum design of the whole design space because it is almost infinite. Hence the problem of modelling systems using this approach is that the design space becomes very large.

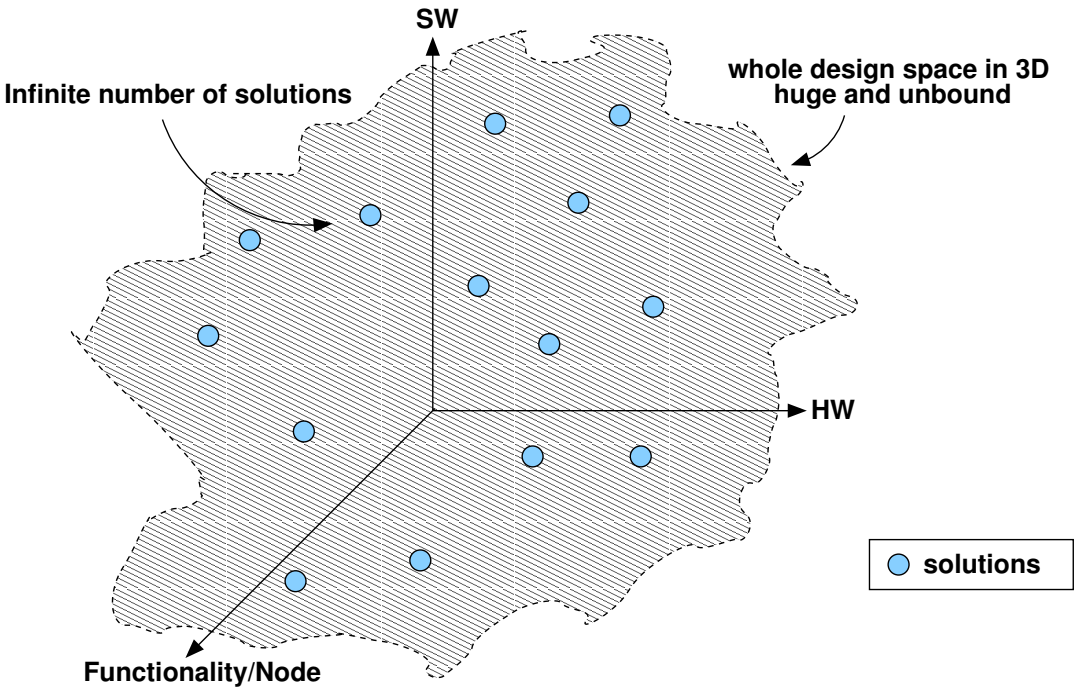


Figure 4.1.: Design space in Naive approach

We start the design process by considering the whole system design space in the architectural exploration. We must explore what functionality will be mapped onto SW/Node and HW/Node, and how SW and HW will be mapped onto Functionality/Node. It is already very difficult to conduct this architectural exploration because the design space, as

mentioned above, is huge. We must explore all of those functionalities in a single step, as shown in Figure (4.2). This is the reason why this approach is more complex and difficult, thus why it is useful to model the systems using the hierarchical approach explained in the next section.

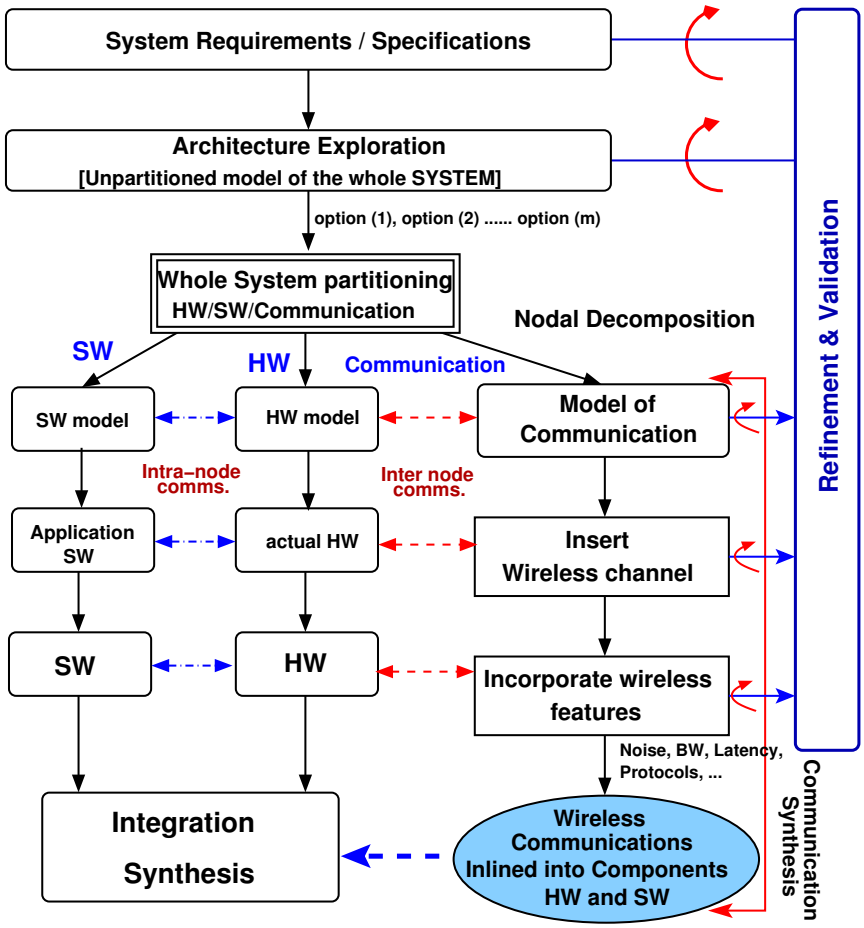


Figure 4.2.: Wireless SystemC design methodology - Naïve structure

4.2.2. System Design Flow Based on Hierarchical Approach

In the hierarchical approach we have HW/Node, SW/Node and Functionality/Node that represent the dimensions of the whole system design space. In the first step of the system modelling process, we need to find the optimum position along the Functionality/Node axis (Figure (4.3-a)); once we have that, we can move along the two other axes (HW and SW), as shown in Figure (4.3-b). This step is known as Nodal composition and it represents the first-level partitioning between Functionality and System. For example, if we have a system containing 20 nodes, each node is modelled with a normal approach represented by second-level partitioning (traditional HW/SW partitioning). The main advantage of breaking the design space into hierarchical layers is to reduce and handle complexity.

Therefore, a hierarchical approach is more efficient and better suited to modelling wireless systems; also, the design space in this approach is limited and smaller than in the Naive approach, as will be described later.

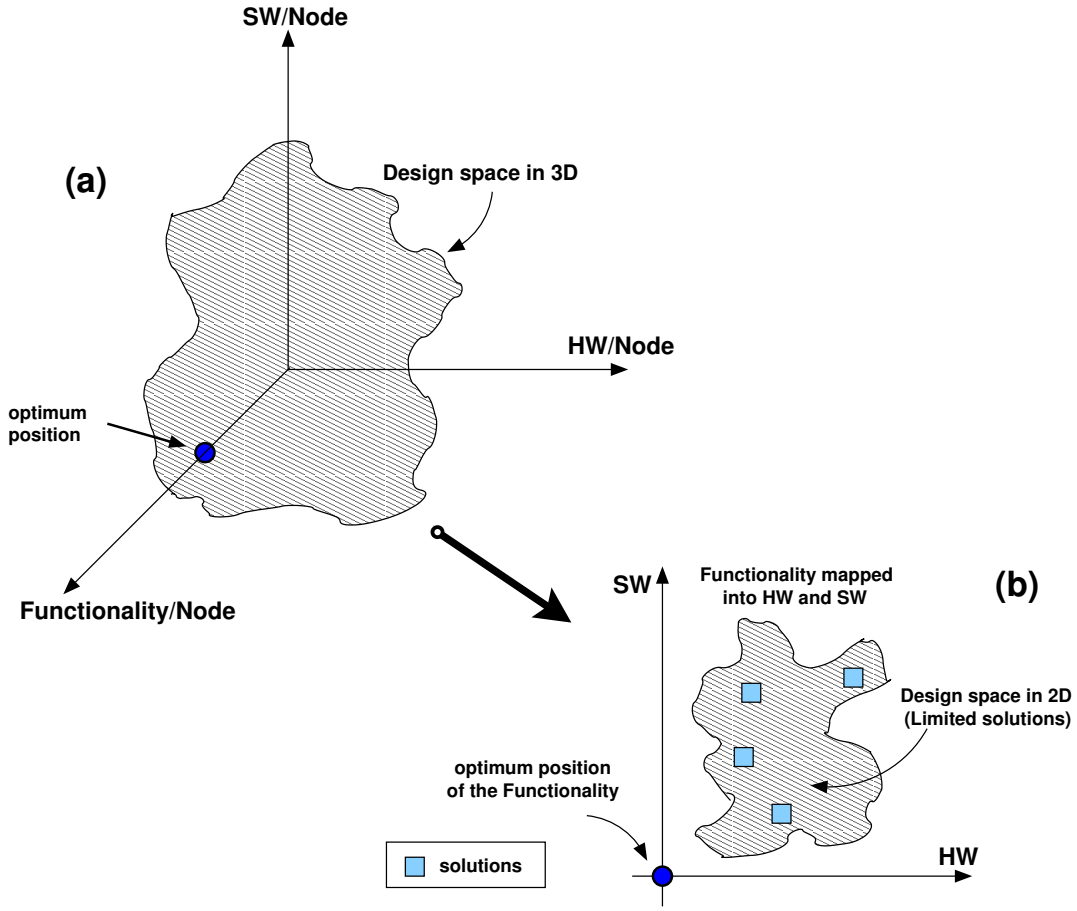


Figure 4.3.: Design space in hierarchical approach

The system design flow of the developed methodology (hierarchical approach) is illustrated in Figure (4.4). This figure shows how Communication alternatives are incorporated into existing SystemC design methodology and how they match their corresponding tasks in the System alternative (HW and SW domains). One part in the design flow of the developed methodology is mainly based on the conventional methodology stages [4], but extra stages are added to describe how wireless communication features are incorporated into conventional SystemC design methodology.

In the first phase of the design flow, the requirement specifications of the whole system are captured and convened into functional and performance requirements. From these requirements, the whole system is constructed as a group of models that represent communication nodes and can interact with each other at a high abstraction level by using SystemC primitive channels such as buffer, FIFO, mutex, semaphore and signal [8]. In this phase, there is no distinction between intra-node communication and inter-node communication because

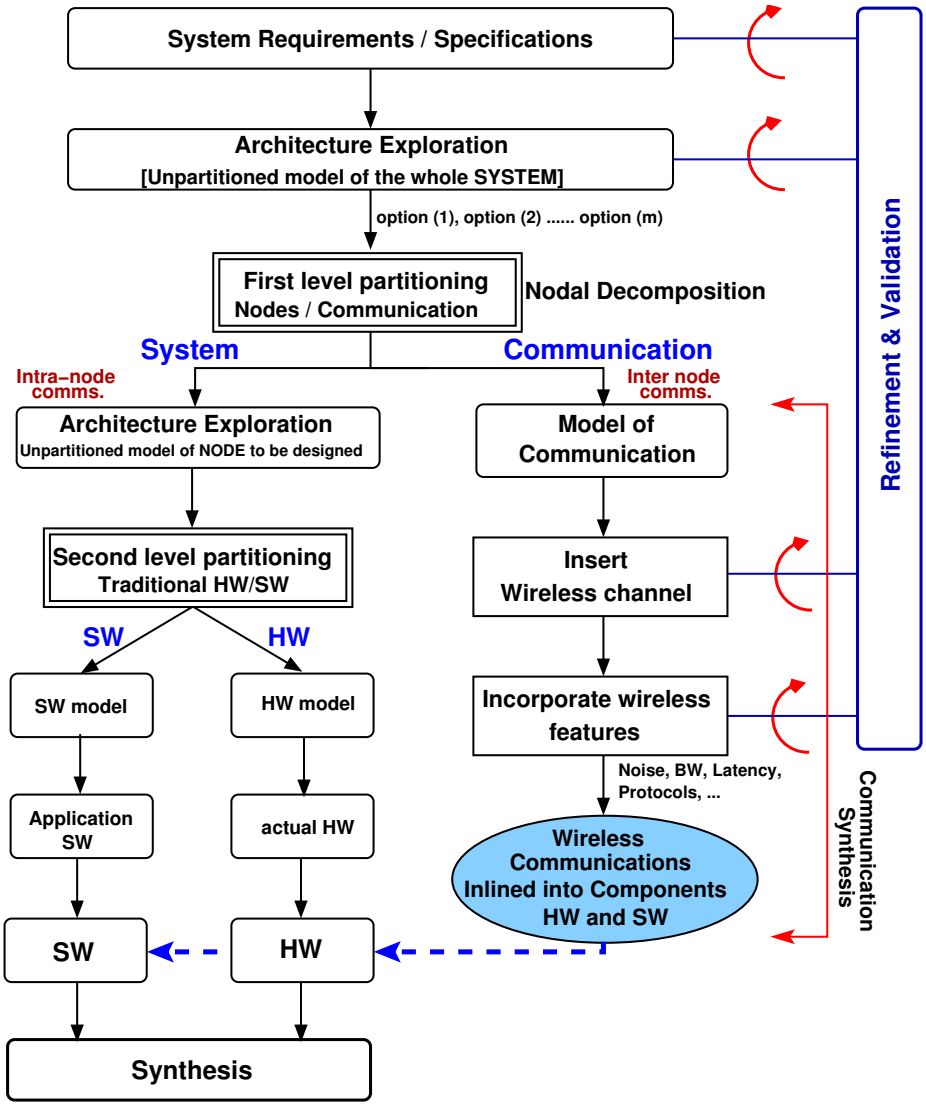


Figure 4.4.: Wireless SystemC design methodology - Hierarchical structure

the specification model does not tell us whether the system is wired or wireless. Both of these options are possible, and the system should be open to architecture exploration that includes all available options. We can say that, for the specification model, we do not need to include wireless attributes at all.

After capturing the specifications, the next phase is architecture exploration of the whole system. The main purpose of this phase is to allocate system functionality to the components [7]. The system architecture is derived from the specifications. More than one option/design can be developed and compared. After comparison, metrics that will be employed to investigate system performance for these options should be estimated; we can then select the best design. Here, the architecture exploration includes the communication elements and the topologies that produce more complexity in the design space. This complexity is reflected in the modelling steps that should help evaluate the optimal partition among hardware or software blocks and communication aspects. Each commu-

nication option can be employed to drive the architecture exploration process and validate the whole system at each refinement step.

After the architecture exploration, the next phase is the first level of partitioning, which is known as Node/Communication partitioning (Nodal decomposition). In this phase the whole system is partitioned into a number of nodes that will communicate to achieve a specific application. This phase also determines whether the function should be achieved by the system board (inside node) or if it can be delegated to communication elements. The Node/Communication partitioning is applied to the whole system model in order to map some communication node models. Here, the node model is referred to as a System model. Communication between the nodes is described as inter-node communication. It is represented by a Communication model (our wireless channel model [133]) that can be used to produce wireless communication behaviour. The next two sections describe Communication and System models.

a- Communication

This first level of partitioning represents inter-node communication, which refers to how the nodes communicate with each other. It is implemented through the wireless channel model. It can be represented at a high abstraction level by a variable (signal, FIFO or shared memory), and can then be refined by inserting a wireless communication model. We can then incorporate wireless features, such as communication protocols, delay, noise, etc.

b- System (HW/SW)

Here, a conventional design flow (conventional SystemC design methodology) can be applied to the System model to be developed. At this point, the second level of partitioning (i.e., which is HW/SW partitioning) must be done. Hardware and software partitioning is done on the node model to map the functionality of hardware or software components based on some constraints such as area, timing, throughput, power consumption etc. It is decided whether the function should be performed by a processor (SW) or by dedicated hardware [14, 52]. The components that are to be modelled as hardware can be refined down to RTL level.

In the last phase of the design flow, hardware and software models are mapped to actual components. HW components can be either synthesised or taken from the market. For SW,

the operating system is defined and the functionality of the SW components becomes application code. The wireless communication model is in-lined into hardware and software components.

c- System Verification

In SoC design methodology, the system design flow process is done after the system specifications are approved. In the system design flow described above, the system behaviour modelling is verified against the functional requirements at each step. We need system verification in order to check the architecture against the intended functional and performance requirements [135].

4.3. Summary

In this chapter, conventional SystemC design methodology is developed to model complex wireless communication systems. The design exploration space is divided into two dimensions: the System (HW/SW) to address exploration and refinement of the system, and the other dimension is Communication to address wireless features. As a result, the integration of communication modelling into design modelling is introduced in the early stages of system development. The methodology is developed by employing a system level model of a noisy wireless communication channel [133] that fulfils SystemC language requirements in order to support wireless systems. For instance, the wireless communication channel model becomes a part of the developed methodology and can then be inserted into any system. We can also introduce some communication delays, communication protocols and noise.

5. Modelling of a Noisy Wireless Communication Channel

The work presented in this chapter introduces the integration of communication modelling into the design modelling at the early stages of system development. It presents the modelling of a noisy digital communication channel using SystemC. It supports different modulation techniques, such as Amplitude-shift keying (ASK), Phase-shift keying (PSK) and Quadrature amplitude modulation (QAM). It supports the setting of different Signal-to-Noise Ratios (SNR) and different types of interference for Point-to-Point (P2P) and Point-to-Multipoint (P2M) platforms. In this work, a demonstrator that implements both P2P and P2M systems has been constructed and incorporated in the whole design methodology. It implements Go-back-N Automatic Repeat reQuest (ARQ) and High-Level Data Link Control (HDLC) to confirm that the channel is working properly.

5.1. Introduction

This work will provide a first step towards this methodology by introducing a simple noisy digital channel in SystemC that can be used to model all system interactions. It will show the construction of a wireless communication system, such the one shown in Figure (5.1), which represents two communication nodes that exchange information through a noisy communication channel. The channel model proposed supports different modulation techniques, such as ASK, PSK and QAM. It supports the setting of different SNR and different types of interference. The system has been checked by implementing a demonstrator, which supports Go-back-N ARQ and HDLC.

The main contribution of this work is the integration of communication modelling into the design modelling at early stages of system development. To our knowledge, this is the first time anyone has undertaken this modelling. The remainder of this chapter describes how the noise is generated. Section three presents the channel module implementation, which

describes the implementation of the noisy digital communication channel by incorporating the noise generation process into the channel. Section four describes the work undertaken on the modelling of Point-to-Point (P2P), Point-to-Multipoint (P2M) and Multipoint-to-Multipoint (M2M) communication systems. Section five describes the data packetization. Section seven briefly describes ARQ protocol. The last two sections describe the simulation results with discussions and conclusions.

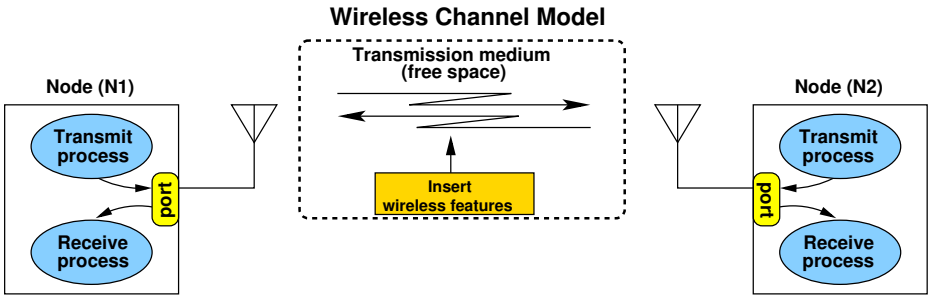


Figure 5.1.: Block diagram of a wireless communication system consisting of two nodes

5.2. Related Work

The modelling of noisy communication channels is not new. Analogue channel modelling using Matlab, Simulink and Opnet is common. For example, behavioural modelling has been proposed in [136], but has not been incorporated into a homogeneous design environment. Work described in [136] is one of the few ones to date about modelling using SystemC. The paper shows a systematic approach to modelling and simulating an Orthogonal frequency division multiplexing (OFDM) transceiver for wireless LAN using SystemC.

5.3. Noise Generation Process

The performance of any communication system is affected by noise and interference from other sources, which play a crucial role in communication systems [72, 88]. In practice, noise represents the number of errors occurring in the system. The simplest model of digital noise is just to consider impulsive noise, in which the individual bits or packets are modified with a given probability [72, 99]. The other type of error is lost bits, which only happens if the system is out of synchronisation. However, it is assumed that there is no model of a Phase Locked Loop (PLL) in the system, and it is assumed to be PLL locked. Consequently, we will not model the second type of error at this stage; it will be

left as an exercise for future work. Here the model is done in a conventional way, and we assume a memory-less system, which implies the Markov property. Therefore, the exponential distribution is a good fit for the model, though other distributions, such as Pareto distribution, can be easily coded as well. Here, for simplicity, we use exponential distribution, but it is not accurate for modelling real world systems and it gives inter-arrival times for errors by solving the Probability Density Function (PDF) of the exponential distribution for t, which represents the inter-arrival time of errors. Therefore, a random experiment is performed to determine the position, where errors are injected in the bit stream.

For each modulation technique there is a waterfall curve, which relates an SNR to a specific BER, as shown in Figure (5.2). The PDF of the exponential distribution is given by $f(t) = \lambda e^{-\lambda t}$ for $0 < t < \infty$, where the $BER = \mu = \frac{1}{\lambda}$ and μ is the mean and variance for the exponential distribution. Therefore, solving it for t to determine the position of errors and introduce them to the channel follows these steps:

1. choosing the system SNR that corresponds to one of the modulation schemes.
2. getting the corresponding BER from the waterfall curves relating the BER to the SNR.
3. calculating $\lambda = \frac{1}{BER}$.
4. performing a random experiment according to the PDF of the exponential distribution.
5. injecting the error in the bit stream.

The process steps (from 1 to 5) of solving the PDF can be easily traced on the graph shown in Figure (5.2).

5.4. Wireless Channel Platform

As mentioned previously, SystemC is a C++ class library and its methodology can be used to model system-level designs and effectively create a cycle-accurate model of software algorithms, hardware architecture and interfaces of SoC (System On Chip) and system-level designs [30, 8]. In this part of the work, structural designs for transmitting, receiving and channel models are implemented in SystemC using modules, ports, processes and signals that represent the fundamental constructs of SystemC libraries. The modules can contain

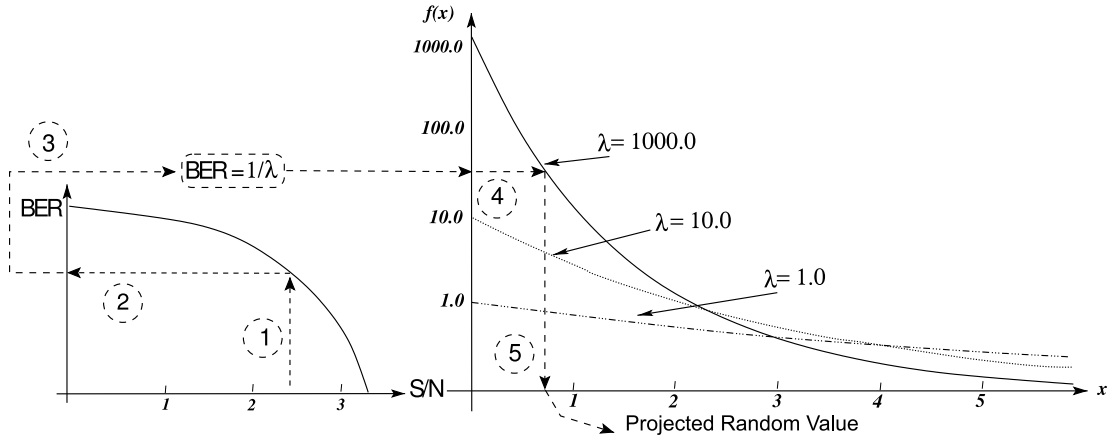


Figure 5.2.: Stochastic projection of noise.

other modules, allowing a hierarchical construction of the system model. The processes communicate with each other via interfaces, channels and ports, and can synchronise with each other via events objects. Also, a variety of data types are supported to include single bits, bit vectors and fixed-point integers [13, 130]. Moreover, SystemC supports several techniques for addressing the complexity of modern designs. Today's system designer can use several approaches for attacking the complexity issues that come with complex system design; these approaches are abstraction, design reuse, team discipline, project reuse and automation [8]. On the other hand, blocks communicate via ports/pins and signals or wires in traditional HDLs. In SystemC, modules are interconnected using either primitive channels or hierarchical channels. Both types of channels connect to modules via ports. The powerful ability to have interchangeable channels is implemented through a component called an interface, which is implemented in this work [30].

5.4.1. Transmitter and Receiver Models

The wireless communication system that is modelled and designed in this chapter, shown in Figure (5.5), consists of one transmitter and one or more receiver and channel, which can be used to create P2P and/or P2M channels. The structural designs for these platforms are implemented in SystemC using modules, ports, processes and signals, which represent the fundamental building blocks of SystemC libraries [8]. The transmitter, receiver and channel models have been implemented using modules and connected through the ports, which are created from the base class `[sc_port <interface[,N]> port_name]` and bound to an interface type [13, 7], as illustrated in the program segment (5.1).

Algorithm 5.1 Transmitter and receiver modules

```
SC_MODULE(transmitter) {
     sc_port<sc_fifo_out_if<sc_bv<N> >,0 > tpackout;
     sc_port<sc_fifo_out_if<sc_bv<N> >,0 >transmitter_data;
...
};
SC_MODULE(receiver) {
     sc_port<sc_fifo_in_if<sc_bv<N> > > rpackin;
     sc_port<sc_fifo_out_if<sc_bv<N> > > receiver_data;
...
};
```

5.4.2. Wireless Digital Channel Model

The noisy digital channel being designed here employs FIFOs as an interface type and elasticity buffer. FIFO is simply a first-in, first-out buffer. Each FIFO has a number of slots for storing values. The number of slots is by default fixed to sixteen during elaboration time [5, 8]. In this system, FIFO's size has been kept with the default value, which is sixteen, because we have found in our simulation experiments that this is more than enough to buffer information packets as they traverse from point to point in the chain. We used FIFOs in order to guarantee that all the packets will remain in the correct order during the transmitting process. For simulation purposes, data packets are represented by binary vector data type and designed based on HDLC [137].

This channel needs to support P2P and P2M scenarios. It has been constructed using modules from the SystemC library and connected through ports, as shown in Figure (5.3). These ports are instantiated from the base class `sc_port` and bound to an interface type [8]. Port interfaces employ FIFO queues as the interface type and are used to connect the system modules to each other. The FIFOs can be accessed using blocking read and write methods. Therefore, synchronisation is implicit, the methods being automatically suspended and resumed, depending on the status of the FIFO channels. This guarantees that no packets get lost [5, 29].

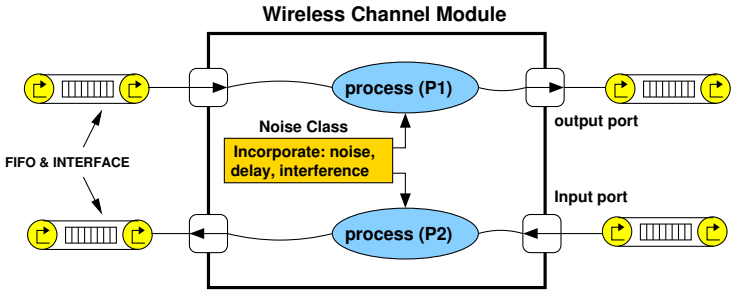


Figure 5.3.: Channel module implementation

5.4.3. Design of Point-to-Point (P2P) Commnication Channel

The basic design of P2P consists of the source module, the destination module, noise class and a channel module, as shown in Figure (5.4). The channel has been implemented as the communication scheme between the source and destination nodes. The source module sends data packets that are created based on the HDLC format through the channel. Then the channel module processes and forwards them directly to the destination. The noise model introduces errors randomly by applying the Stochastic technique, which is used to simulate the error event [138]. The occurrence of the error event is dependent on the selected distribution, which will be related to the error rate. On the other side, the receiver module receives data packets with some errors from the channel module and then sends them to the monitor module. Finally, the monitor module is used to analyse the packets in order to detect the errors and evaluate the error rate by comparing the packets sent from the transmitting module with the packets received by the receiving module. The process continues until all data packets are sent [135].

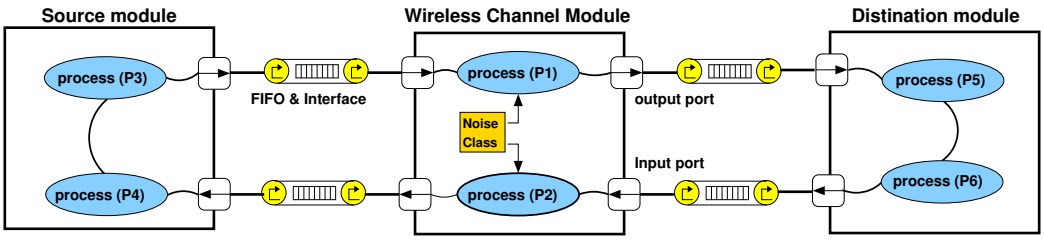


Figure 5.4.: Simple point-to-point communication channel structure

5.4.4. Point to Multipoint (P2M) Construction

P2M is more complicated than P2P, because in this case we need to define multiports and set different noise values and communication parameters for each channel module. Multiports are created using the port_array feature of SystemC and by providing a second parameter *N* in the base class `[sc_port <interface[,N]> port_name]` and then binding them to an interface type [7, 8]. We assume that the source node is connected to *N* target nodes via *N* interface channels; therefore, the receiver modules are assigned a position in the array to connect the channel modules on a first-come first-served basis, as shown in Figure (5.5). There are two ways of connecting; the first is known as a shared channel and the other one as multiple channels. The first method assumes a schematic of master slave synchronisation mechanism, such that only one slave is allowed to transmit in the feedback channel at any given time. For the general case it is required to model the contention whilst accessing the same resource. In the second method, to create the

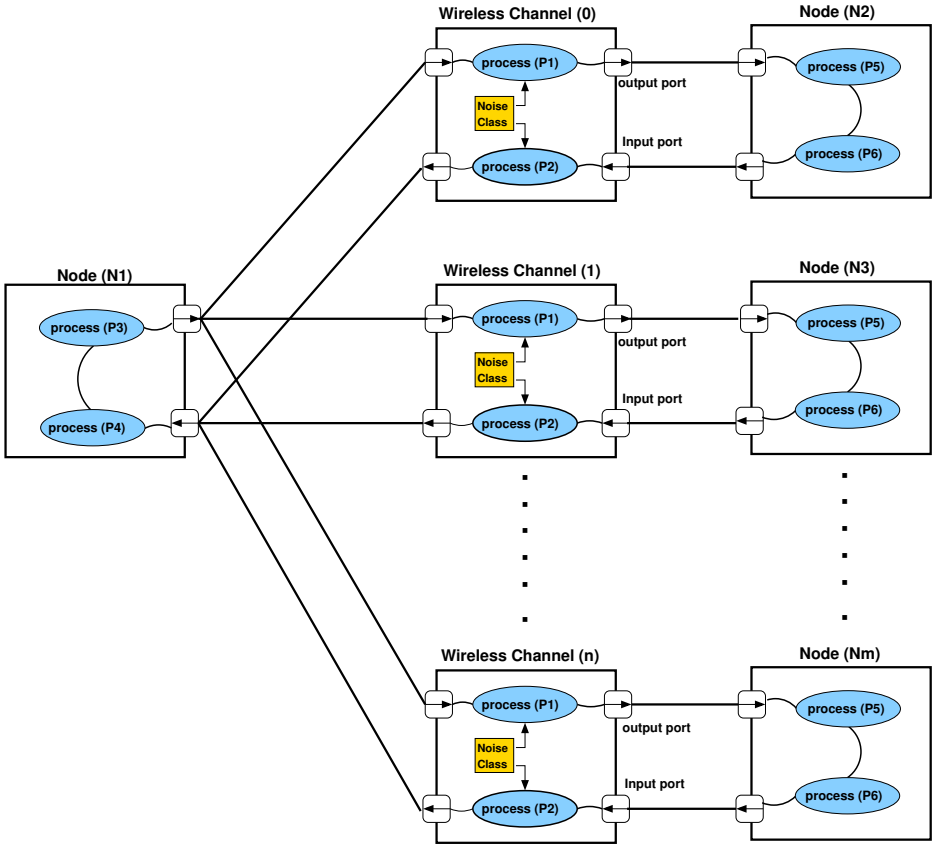


Figure 5.5.: Point-to-Multipoint communication channel structure

P2M platform all modules are first created during the elaboration time; namely, the source module and the N channels and N destinations are dynamically instantiated. Secondly, the module interfaces are instantiated. Finally, the named port binding of the modules and interfaces is applied between the source module and the channel modules, as well as between the channel modules and destination modules. In this work, the latter is selected to describe P2M, because it assumes there is no correlation between channels. Moreover, this structure fits well with multiport definition and the module structure of SystemC [5, 22].

In the code segment (5.2), lines (10-14) show how we can create N transmitter modules by using dynamic instantiation features. Moreover, by using this feature of the modules we can generate as much as we want from these devices (models). After that, all the modules that have been instantiated have to bind together. Lines (19-22) illustrate how the transmitter module has been bound to the channels; in the same manner, a binding process is applied between receiver modules and channel modules, as shown in lines (24-27). Finally, we have to note that dynamic instantiation and binding must be done during elaboration of the simulation [5, 7, 8, 43]. Figure (5.5) shows the block diagram of the P2M channel platform.

Algorithm 5.2 Implement P2M platform

```
SC_MODULE(top) {
SC_CTOR(top){
...
transmitter *Transmitter[NTransmitters];
receiver *Receiver[MReceivers];
channel *Channel[MChannels];
compare *Compare[MChannels];
...
// Generate (N = NTransmitters) Transmitters
for (unsigned i=0; i < NTransmitters; i++) {
     std::ostringstream trans_name;
     trans_name << "Transmitter" << i << std::ends;
     std::string s = trans_name.str();
     Transmitter[i] = new transmitter(s.c_str());}
...
...
//this for loop to bind the Transmitter to Channels trough TCh_fifo
Transmitter[0]->clock(Tclock);
for (unsigned i=0; i< MChannels; i++){
//bind fifos to module ports
     Transmitter[0]->tpackout(*TCh_fifo[i]);
     Transmitter[0]->transmitter_data(*TComp_fifo[i]);}
//this for loop to bind the Receivers to ChR_fifos and Receivers to
RComp_fifos
for (unsigned i=0; i< MReceivers; i++){
Receiver[i]->clock(Rclock);
     Receiver[i]->rpackin(*ChR_fifo[i]);
     Receiver[i]->receiver_data(*RComp_fifo[i]);}
...
}//end of constractor
};
```

5.4.5. Multipoint-to-Multipoint (M2M) Construction

At this first stage of the implementation phase, our goal is to construct a channel model that can be used to support contention and noncontention based shared wireless channel access, and includes P2M and Multipoint-to-Multipoint (M2M) communication scenarios. The wireless channel module is responsible for carrying the data packets to all stations, and represents our communication medium in this work. The wireless channel module, as shown in Figure (5.6), models the communication medium. It behaves like a multi-tap bus, where the multiple nodes are connected through it. The behaviour of the link has been modelled as a wireless communication medium.

In this scenario an external object called a ‘mutual exclusion lock’, or ‘mutex’, has been

used to control access to the shared medium. The behaviour of a mutual exclusion lock is used to control access to a resource shared by concurrent processes. A mutex will be in one of two exclusive states: unlocked or locked. Only one process can lock a given mutex at one time. Whenever a node has a packet to send, it tries to lock the mutex. When it locks the mutex it has access to the channel; the rest of the nodes have to wait until the node has released the channel. The other nodes may be subsequently locked by the mutex [7, 8]. This scenario is covered in detail in Chapter Nine.

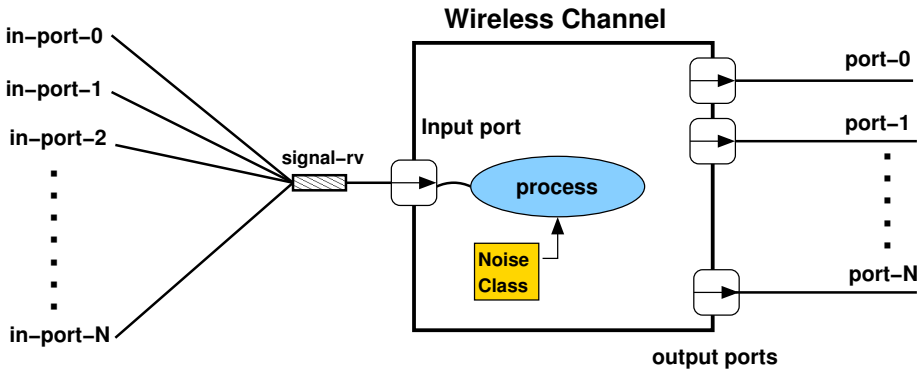


Figure 5.6.: Multipoint-to-Multpoint communication channel

5.5. Data Packetization

In the data transmission process achieved in this system, there is no mechanism to mark the beginning or end of a packet, so by using HDLC technique, the beginning and end of each packet must be identified. This is done using a packet delimiter, or flag, which is a unique sequence of bits that is guaranteed not to be seen inside a packet [137]. In this system, we have used Asynchronous Balanced Mode (ABM), which combines station may use to initiate a transfer [139].

5.6. ARQ Communication Protocol

Automatic Repeat reQuest (ARQ) is an error-control approach that can be used for data transmission. It uses acknowledgements and timeout signals to perform reliable data transmission over an unreliable service [22, 140]. For example, if the transmitter does not receive an acknowledgment before the timeout, it usually retransmits the packet until the transmitter receives an acknowledgment or exceeds a predefined number of retransmissions. This means that, at any time, if the transmitter detects an error when ex-

changing data with the receiver, specified packets have to be retransmitted [141]. There are three types of ARQ protocols, as follows:

- Stop-and-wait ARQ
- Go-Back-N ARQ
- Selective Repeat ARQ

These protocols exist in the Data Link or Transport Layers of the OSI model [141].

5.6.1. Stop-and-wait ARQ

In a backward error control system, a packet is transmitted by a transmitter which then waits for the receiver to indicate that it has received this packet before transmitting the next packet. If the packet is corrupted or lost, no acknowledgement will be sent by the receiver. The transmitter will wait for certain period of time. If the acknowledgement does not arrive within this time, the transmitter then re-transmits the previous (lost) packet and waits again for an indication of its receipt. This type of data link protocol is known as a "stop and wait" ARQ, or idle RQ [140, 141, 142]. As above is referred to as an Automatic ReQuest protocol, where there is not always a requirement to wait for the receipt of the ACK associated with a packet before transmitting the next packet. Idle RQ protocol is just one specific example of an ARQ protocol - one which is quite inefficient as considerable time is spent waiting for data rather than transmitting.

5.6.2. Window Flow Control

To overcome the inefficiency associated with idle RQ, it is possible to transmit new packets up until a given limit without waiting for any received ACKs [22, 143]. The number of outstanding unacknowledged packets is known as the window and is fixed to some maximum value, and leads to the idea of a sliding window protocol. Idle RQ protocol is often referred to as a sliding window protocol with a window size of 1. Use of windows in this manner provides a degree of flow control and prevents a transmitting node overwhelming a receiver (quite likely in the case of re-transmissions), as well as ensuring an upper limit on the buffer space required [140, 142].

Therefore, to deal with the lost packets, for every packet transmitted, the transmitting node maintains a timer. If a particular timer lapses and an acknowledgement has not been received, the transmitter assumes that the packet is lost and that another has to be

re-transmitted [141]. Alternatively, breaks in sequences can be used to indicate that a packet has been lost. There are then two re-transmission options available. Either the node transmits only the packet that was in error, this approach is called selective re-transmission; or it also re-transmits all the packets transmitted after the lost packet, this method is called Go-Back-N transmission [22, 141, 142]. This work simulates the latter approach.

Selective Repeat ARQ

In a selective scheme, the node transmits only the packet that has errors. This has the advantage of being less wasteful of link capacity, but relies on the receiver being able to ensure that packets are delivered to the network layer in the correct order. In sliding window protocols, there is a number called a sequence number; its role is critical [22, 142]. As a packet is transmitted it is given a number identifying when it was output in relation to other packets. These numbers are used by the receiver to ensure that the packets are passed onto the network layer in the correct order [140, 141].

The range of sequence numbers allowed is a function of both window size and retransmission method. In the case of a sliding window protocol with a window size of 1, only a one bit sequence number is required [141]. In general, with a window of K packets, the range of sequencing numbers required is 0 to 2K+1 for a system utilising selective re-transmission and 0 to K+1 for Go-Back-N systems. In practice, sequence number fields within packets are preset, usually to 3 bits or 8 bits.

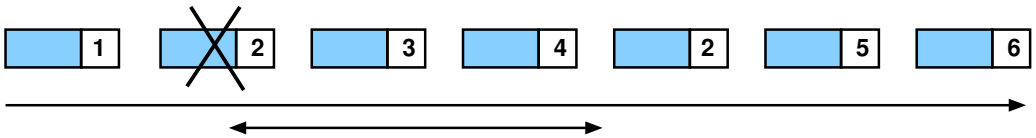


Figure 5.7.: Selective re-transmission

Go-Back-N ARQ

In a Go-Back-N scheme, which is modelled in this work, the transmitting node also retransmits all the packets that were transmitted after the lost packet. Go-back-N retransmission would be used when no re-sequencing could be done, and the retransmission order would be preserved at the cost of lost capacity [140, 142, 143]. In a Go-Back-N system the receiver uses sequence numbers to identify when an error has occurred or when it has received duplicate packets. If a packet has been received correctly it will have a sequence

number N, and the next packet received should have sequence number $N+1$. If the number received/detected is $N+2$, a packet has been lost [22, 141].

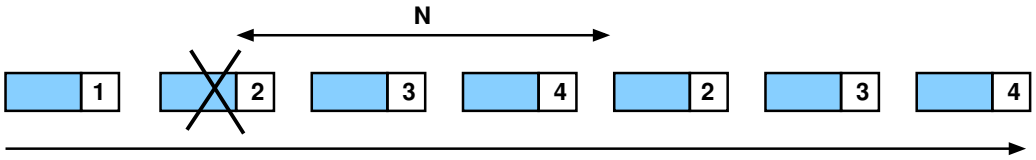


Figure 5.8.: Go-Back N re-transmission

5.7. Simulation Platform

In SystemC, functional verification is done through simulation by applying stimulus to the Device Under Test (DUT) and verifying the response against an expected result [135], as shown in Figure (5.9). In order to test and evaluate the noisy digital channel described in this work, a test platform consisting of the P2P design described earlier using Go-Back-N ARQ and a packet format based on HDLC has been constructed, as described in [22]. The stimulus sends a bit stream to the source module. The bits can then be sent across the noisy digital channel to the destination module. The monitor module gets these bits from the destination module in order to test the channel module, i.e. it receives the bits with some errors from the channel module and then checks, verifies and analyses the bits in order to detect the errors. The modelled system is assumed to be in a clock locked state. So there is no need to model a Phase Locked Loop (PLL) for clock recovery.

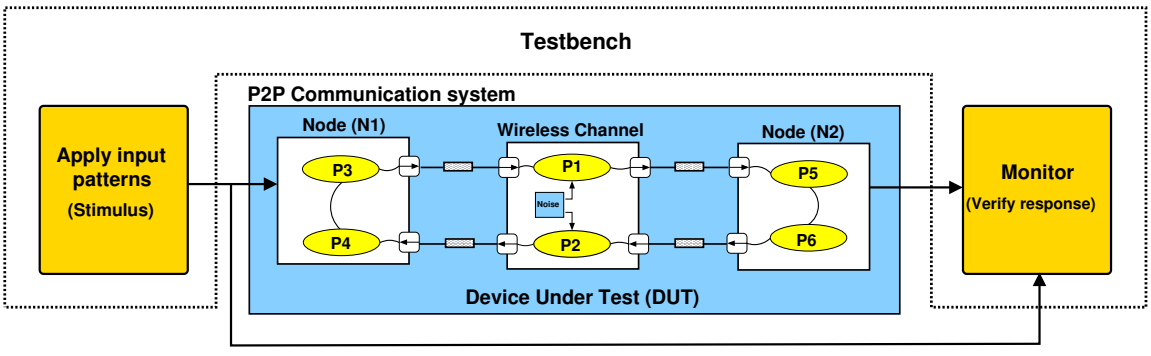


Figure 5.9.: Simulation platform

5.8. Results and Analysis

Since a simulation can process only a finite number of bits or packets, the *BER* needed for evaluation can only be determined depending on the number of packets [93]. In this work, the *BER* is determined by passing a large number of packets through the system and counting the errors at the destination, i.e. packets with errors observed at the monitor module divided by the total number of packets sent from the source. We have set our simulation to run until the *BER* converges. In the simulation of P2P, we set the *BER* to 0.02; the result of replicating the random experiment of passing a large number of packets or bits through the random channel is shown in Figure (5.10). A *BER* based on any number of transmissions gives a spread of results, which are evaluated when an error occurs. This spread is related to the variance of the estimated value in general. In order for simulation results to be useful, the spread should be small and converge to the given value of *BER*. Note that, for the results shown, the variance grows smaller as the number of packets injected grows larger. This is typical behaviour for a correctly developed estimator. Figure (5.11) compares the total number of packets received over a noisy wireless channel against packets successfully received and packets with an error.

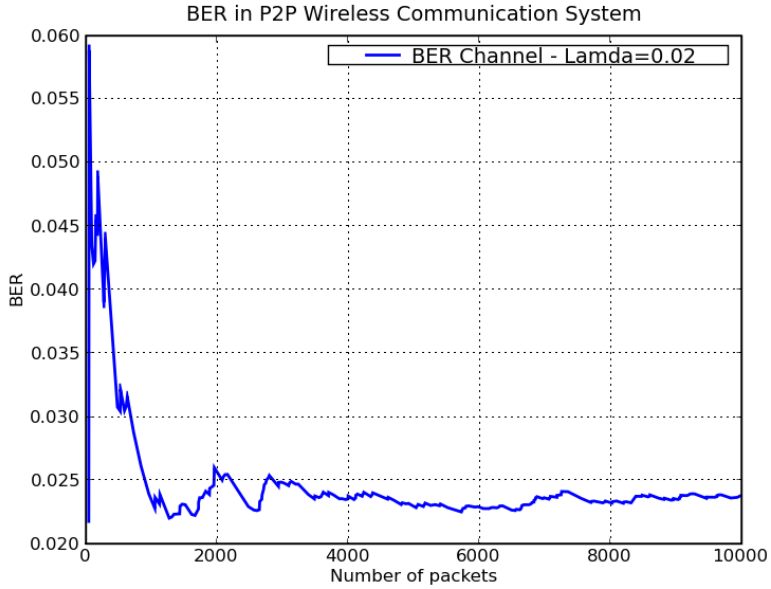


Figure 5.10.: Changing in *BER* versus number of packets in P2P communication channel

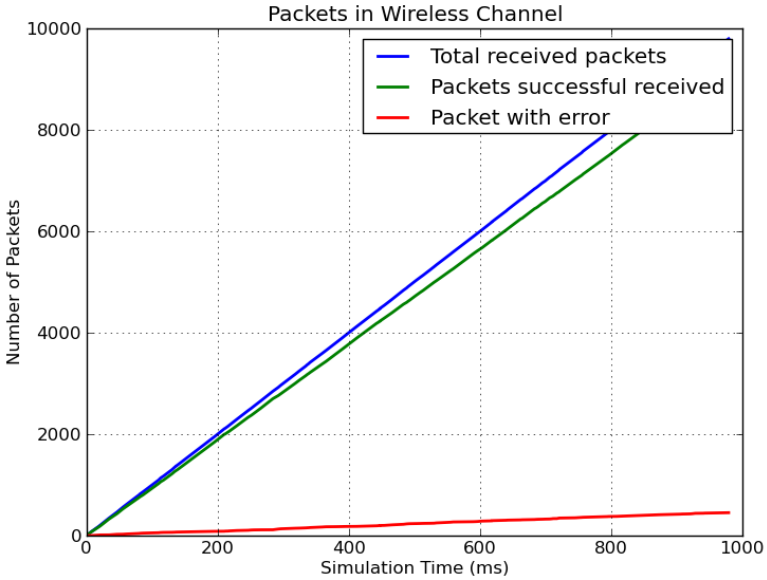


Figure 5.11.: Packets in the noisy communication wireless channel

For the P2M platform we have set $10 \leq \lambda \leq 10000$ randomly to ensure different values of λ; also, each channel is set to process each data packet with different noise. In this simulation, we obtained $\lambda_1 = 8400$, $\lambda_2 = 7800$ and $\lambda_3 = 9100$ with number of packets $N = 10,000$. Figure(5.12) shows the P2M *BER* results with three channels and three receivers.

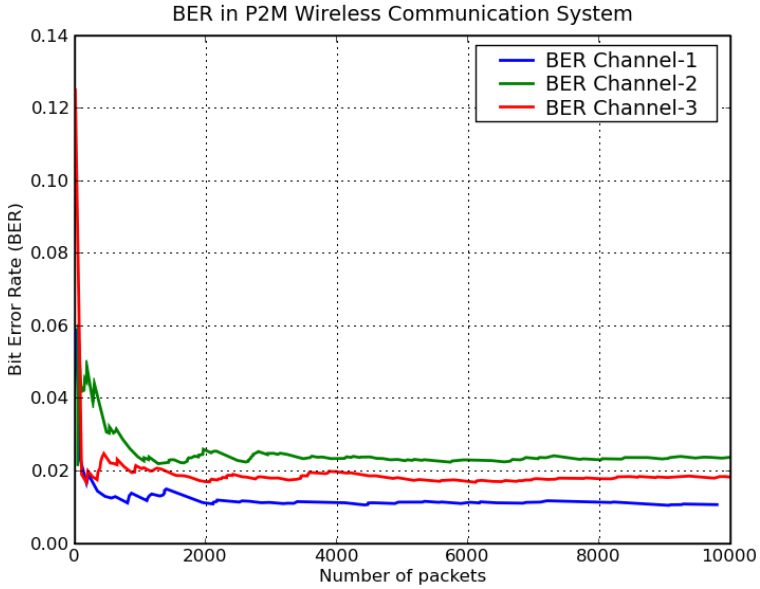


Figure 5.12.: Changing in *BER* versus number of packets in P2M communication channel

We have seen that the noisy channel has been simulated correctly and our platform has

verified the correct simulation of the system and the correct injection of errors in the bit stream. A sample of the expected throughput for P2P (shown in Figure (5.13)) and for P2M (shown in Figure (5.14)) has been determined by the monitor with an ARQ window size set to 7.

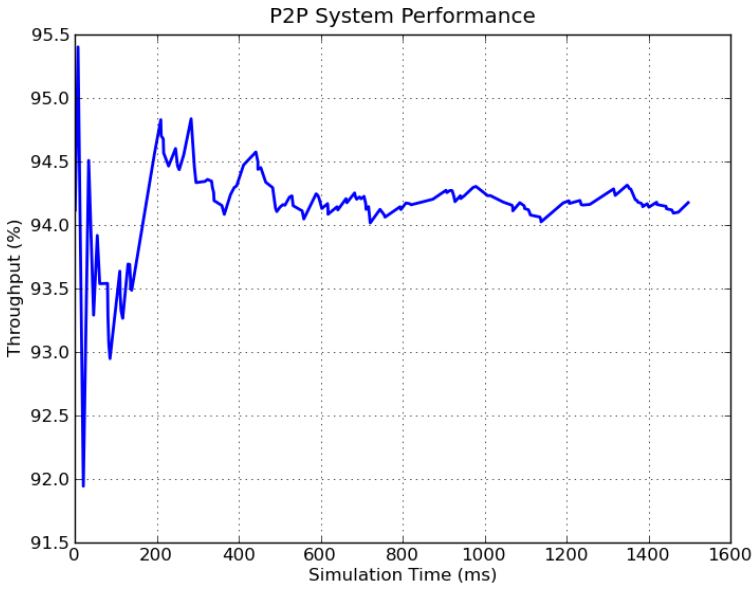


Figure 5.13.: System throughput for P2P with ARQ window size 7

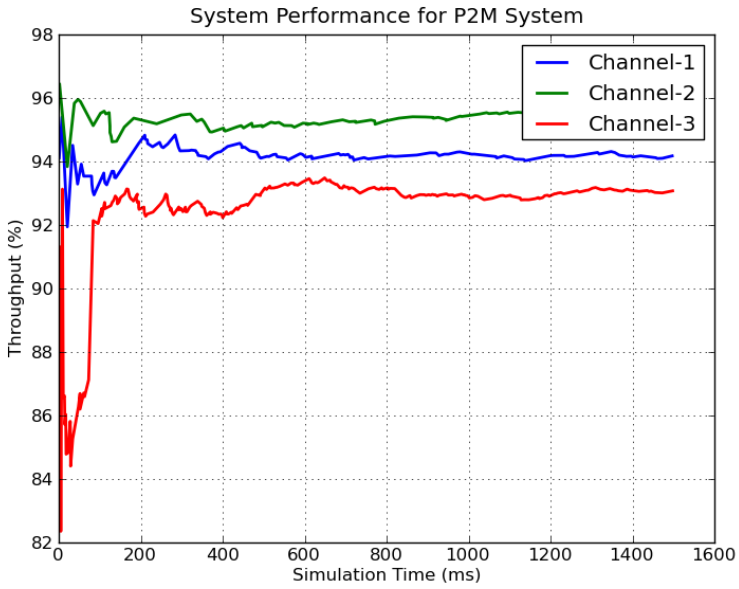


Figure 5.14.: System throughput for P2M with ARQ window size 7

5.9. Summary

This work presents a first step towards the integration of communication modelling into design modelling at the early stages of system development. The simple noisy digital channel can be used to model all communication system interactions. This work demonstrates a simple and computationally efficient way to model a communication channel within a system level. The model is developed at a high level of abstraction, allowing for rapid simulation and early estimation, which are necessary for successful system development using the SoC design methodology. SystemC has been chosen because it provides a homogeneous platform for the design and modelling of complex systems. Furthermore, as systems become more tightly integrated, the ability to evaluate system performance at early design stages becomes increasingly important. This is facilitated by the SystemC design methodology and by following an IP-based design. To our knowledge, this is the first time that the modelling of a wireless communication system has been undertaken in SystemC and incorporated into a uniform design methodology, suitable for developing new technologies following the SoC design methodology.

Part III.

Case Study: Flocking Behaviour System

6. Modelling of a Flocking Behaviour System Based on Shared Variable Communication

At this stage of the work, a demonstration must be created by developing a small application and/or test case in order to validate the developed methodology. The application is known as a flocking behaviour system[18]. By modelling this system, we need to prove that incorporating and fixing the wireless channel, wireless protocol and/or noise early in the design methodology is very advantageous, i.e., small changes in the wireless specifications will create big changes in the system dynamics. Therefore, the system might be constructed in different ways to investigate the system over different performance parameters.

The wireless communication of this system is created through the use of a wireless channel model in two scenarios. The first is P2P, meaning that there is a channel between every two particles. In the second scenario, communication between the particles is based on the shared channel. In the P2P scenario, the approach covered in this chapter for the communication is modelled at a high abstraction level using shared variables. The communication is then refined by inserting a noisy wireless channel; this step will be covered in the next chapter. The second scenario, where communication is based on the shared channel, will covered in Chapter Nine.

6.1. Informal Description of the Flocking Behaviour System

Flocking is the term given to a group of particles behaving in a particular way. Each particle is modelled individually and moves at its own velocity [123, 107]. At any time, each particle is defined by its position in space and its velocity vector. The velocity of each particle is updated based on the parameters of the surrounding particles. Even though the

interactions are only local, the flock presents a collective behaviour. In order to maintain the system stability, all particles should be kept close to each other without colliding. In this work, the flocking behaviour system is modelled and designed based on Reynolds' three principles [112]. These are:

- **Separation:** This refers to collision avoidance. The particles always try to steer away from other particles near them.
- **Cohesion:** This is when particles move toward the average position of local flockmates.
- **Alignment:** This refers to velocity matching. The particles must match their velocity to that of other particles.

At any given moment, a particle only detects other particles located within its surrounding region. The movement of each particle is evaluated depending on the error value calculated based on the relative position and absolute position of the particles located within its radius of perception.

6.2. System Definition

Consider a flocking behaviour system of one leader and $N-1$ particles followers. All the particles, including the leader, are operating in the same workspace $W \subset R^2$. Let a particle $P_n = (X_n, Y_n) \in R^2$ denote the position of particle P_n as shown in Figure(6.1). The configuration space is spanned by $P = [P_0, ..., P_{N-1}]$. Each of the N particles has a specific coordination (X_n, Y_n) with respect to the global coordinate system [124, 144, 145]. The configuration of each particle is represented by $P_k = (X_k, Y_k) \in R^2$, $k = 0 to N-1$.

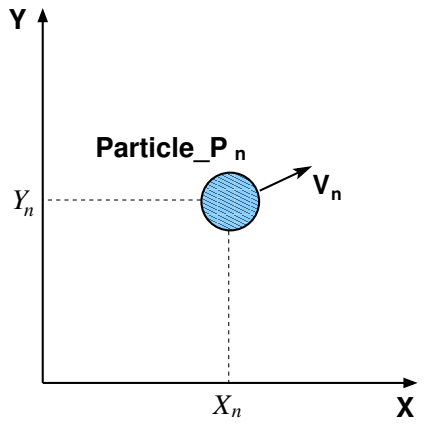


Figure 6.1.: The representation of a particle

Each particle is designed to converge to a desired relative configuration with respect to a certain subset of the rest of the group, in a manner that will lead the whole system to a desired formation. Specifically, each particle (P_i) is assigned a specific subset (N_i) of the rest of the group, called particle i's communication group, with which it can communicate in order to achieve the desired formation.

6.3. System Model

This system has a large number of particles distributed in the environment. The particles use wireless communication to communicate with each other and determine each others' location. They should maintain stability and must stay together. They are controlled by a leader and should converge in a certain area and be distributed around the leader's position. In different scenarios, they can follow the leader to a specific position.

In terms of construction, there are many ways to connect the particles. In this chapter, the system is investigated by constructing ring topology and fully connected topology. Different forms of architecture must be explored in order to investigate the system over different performance parameters. In a ring topology, each particle can communicate only with its neighbours (Figure (6.2-A)); it can send and receive packets over a dedicated channel. At this stage, the channel is an abstraction of the transmitting medium, and connects just two particles; it is a point-to-point channel. The other topology is fully connected (Figure (6.2-B)). Here each particle is linked to all other particles in the flock. In other words, each particle can communicate with the other particles, creating the flock system. Particle P_0 is controlled and selected as the leader. In this stage, the system is constructed at a highly abstracted level of communication, i.e., communication between particles is done using shared variables.

6.3.1. Flocking Control System

The control algorithm of the whole system is linked to the actual transmission and implementation. Here, the control refers to the stability of the system. It is represented by a proportional and derivative controller (PD controller), as shown in Figure (6.3). The main function of the controller is to update the acceleration and velocity of the particles using the current position of each particle and the error value that is evaluated based on the data collected in the previous clock cycle, as shown in Equations (6.1) and (6.2).

$$x_{acceleration} = \varepsilon_x * K_P + (\varepsilon_x - old\varepsilon_x) * K_d \tag{6.1}$$

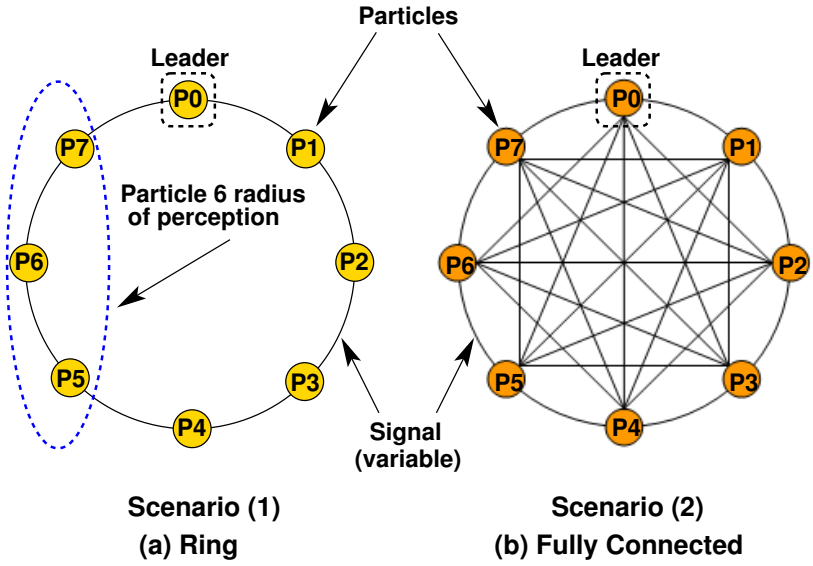


Figure 6.2.: Flocking behaviour system constructed at a high abstraction level

$$y_{acceleration} = \varepsilon_y * K_P + (\varepsilon_y - old\varepsilon_y) * K_d \tag{6.2}$$

where:

ε_x: current error in x direction.

ε_y: current error in y direction.

K_P: proportional gain.

K_d: derivative gain.

$old\varepsilon_x$: error from the last cycle in x direction.

$old\varepsilon_y$: error from the last cycle in y direction.

$x_{acceleration}$: current acceleration in x direction.

$y_{acceleration}$: current acceleration in y direction.

6.3.2. Communication Between Particles

'Communication' refers to how the particles will communicate. At this stage, the particles communicate at a highly abstracted level. The particles can 'know' the position of their neighbours by communicating messages. In both topologies mentioned above, each

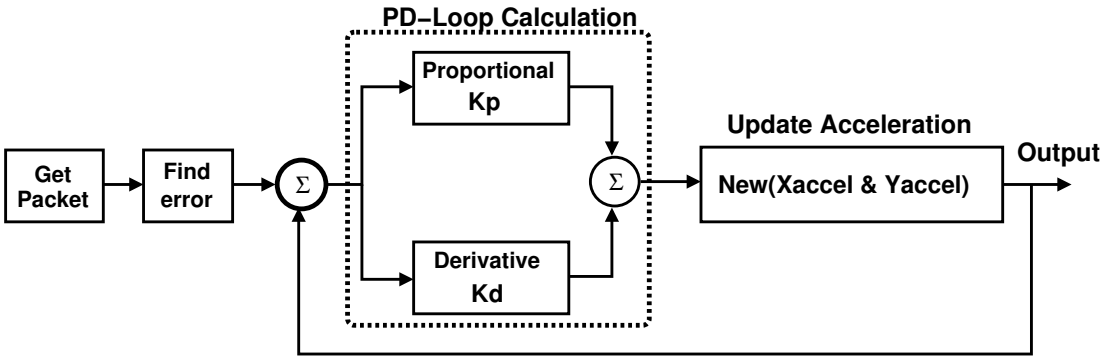


Figure 6.3.: Particle control system

particle can communicate (by transmitting and receiving messages) with the neighbours located on its radius of perception, as shown in Figure (6.2). In a system constructed based on ring topology, Figure (6.4) shows how the particles can exchange messages. Particle P_5 can send its position to particle P_4 and particle P_6.

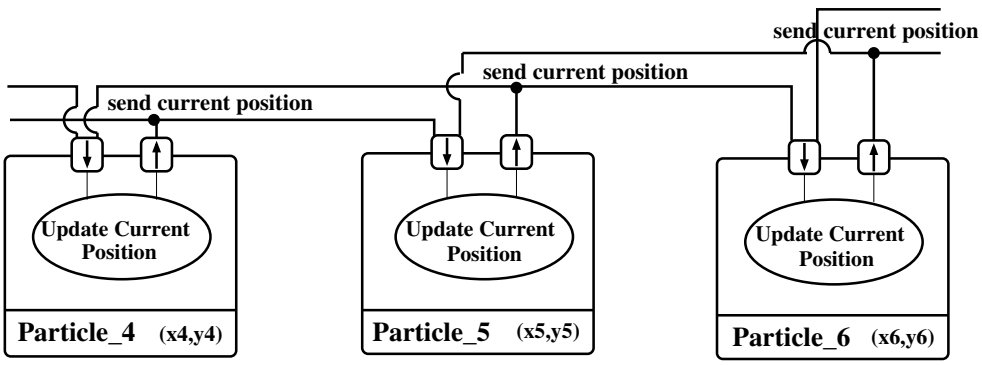


Figure 6.4.: Transmitting and receiving communication messages between particles

6.3.3. Particle Module Structure

The particle module is constructed based on two processes: transmitting and receiving. Each particle performs a simple task, which is transmitting its current position to the other particles located on its radius of perception (as shown in Figure (6.2)) and receiving other particles' positions. The particle module consists of three main elements: a transmitter process that can be used to send data to the other particles and the convergence module, a PD controller used to move the particles to new positions, and an estimation process to evaluate error. The particle module structure is illustrated in Figure (6.5). It has the following ports:

- An input port to send packets.
- An output port to receive packets.

- An output port to send errors to the convergence module.

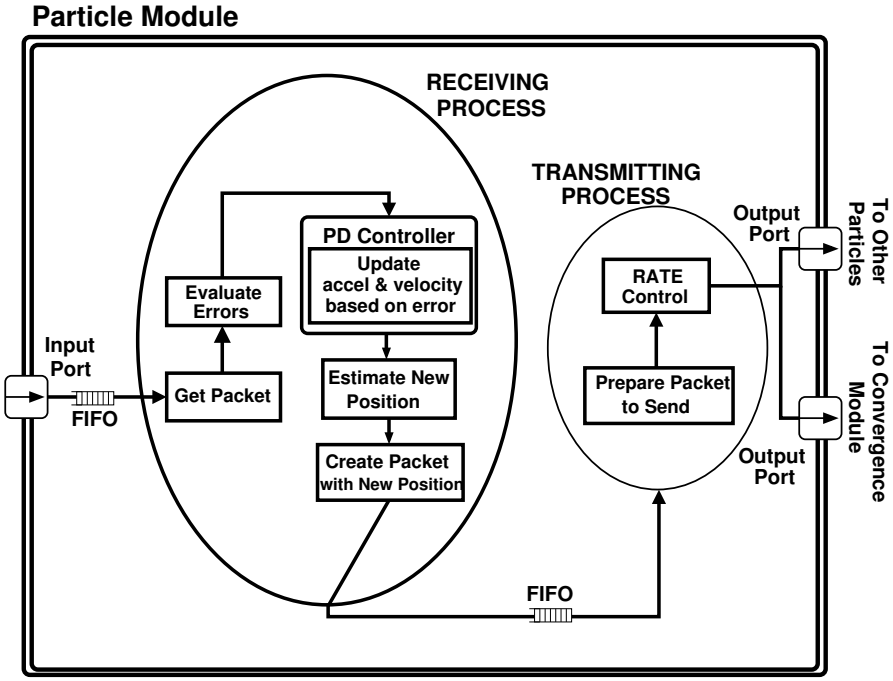


Figure 6.5.: The structure of the particle module

6.3.4. Convergence Module

The convergence module is designed to receive error values from the particles in order to evaluate the convergence point, and can then be used to stop simulation at this point. Since all the particles send their error values at the same time, the convergence module designed here employs FIFOs as an interface type and elasticity buffer [30, 8]. FIFO is, simply, a first-in first-out buffer. Each FIFO has a number of slots for storing values. The number of slots is by default set to sixteen during the elaboration time. In this system, FIFOs have been used to buffer error values received from all particles, because to reach the optimum convergence point, we must guarantee that all error values are received and will remain in the right order during the transmitting process. As shown in Figure (6.6), the convergence module structure has only one input port which is used to receive error values from the particles, and one process employed to find the running average of the error values and check the convergence point of the whole system.

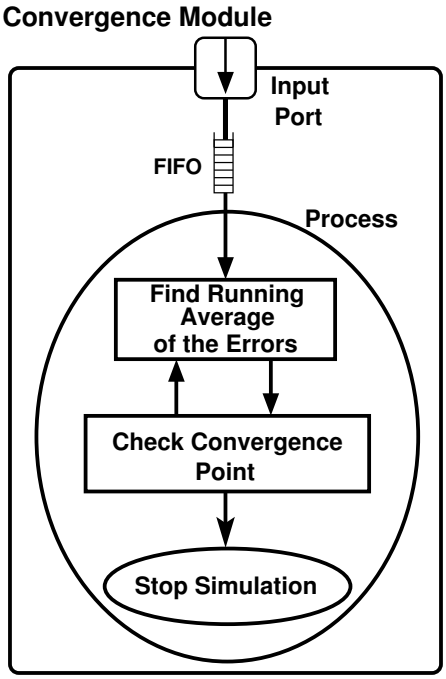


Figure 6.6.: The structure of the convergence module

6.3.5. Data Packet

The packet is the unit of data exchanged among particles during the simulation. Its size has the most profound affect on the number of packets sent across the system (between the particles), number of particles and relative position. To determine the best packet size for our system, some empirical testing is required. Its format is shown in Figure (6.7). It consists of the following fields:

1. **Packet number:** This represents the total number of packets transmitted by each particle.

2. **Particle number:** This is the particle ID; it has a value from 0 to $N-1$, where N is the number of particles used to construct the flock.

3. **Relative position:** Each particle's relative position has to be defined at the beginning of the simulation, because the particle should have reached this position at the end of the simulation.

4. **Absolute position:** This defines the x and y coordinates of each particle with reference to the starting point $(0,0)$.

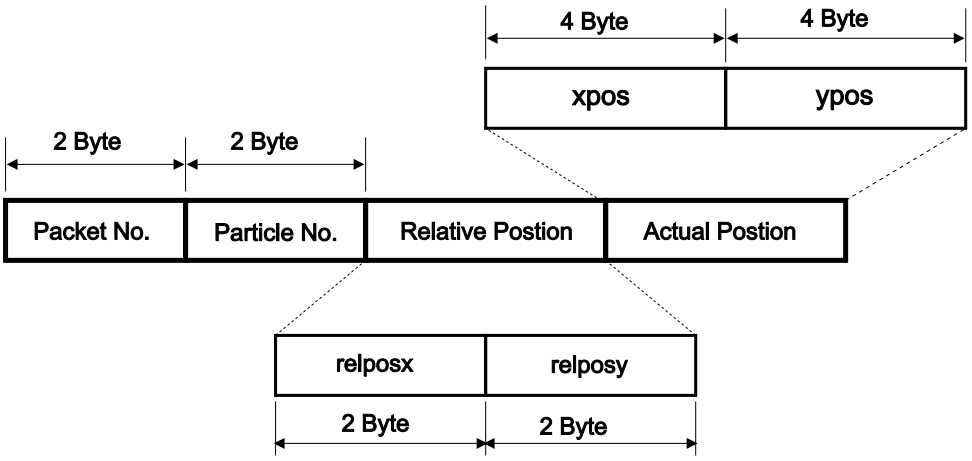


Figure 6.7.: Data packet format

6.4. Simulation Platform

In SystemC, the functional verification of the modelled system is performed through simulation. This process consists of applying a stimulus to the Device Under Test (DUT) and verifying the response against an expected result. At each time step in the simulation, the operations shown in Figure (6.9) are applied to all particles simultaneously, and the positions and velocities of all particles at the next time step are updated accordingly. Subsequently, each particle must send its updated position to the other particles located within their radius of perception.

6.4.1. Initial Conditions

Various initial conditions may be considered for our modelled system; these conditions are as follows:

- **Relative position:** The particles are distributed uniformly in space, with a defined shape. The shape is created based on the relative position value given to each particle at the beginning of the simulation; the distance between the relative positions represents the desired distance (L_d) that should be maintained by each particle in order to make the system conform to the ideal pattern shown in Figure (6.8). At the final position, the system has reached the convergence point if each particle maintains the desired distance from the particles located on its radius of perception.

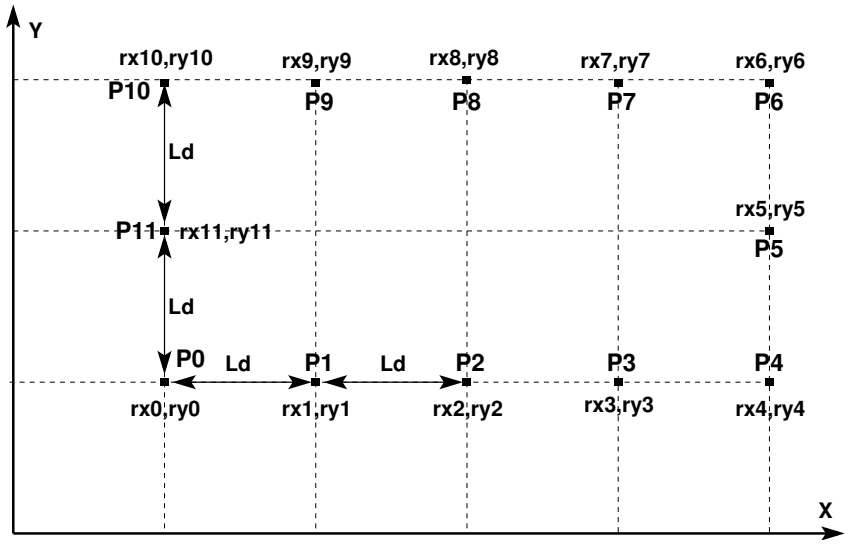


Figure 6.8.: Relative position for N=12

- **Single flock:** We start our system by creating a single flock. The topologies under investigation in this system are the ring and the fully connected, as mentioned above.

- **Set leader:** At the beginning of the simulation, we must select one of the particles as the leader. If the leader moves, all the particles should follow; otherwise, (leader-fixed) all the particles should keep close to each other without colliding and attempt to conform to the desired pattern around the leader based on their relative positions. In this work, P_0 is selected as the leader.

- **PD Controller:** The most important parameters for the flocking behaviour system are the controller parameters (K_p and K_d). The values of these constants affect the way particles will interact with each other because the controller is responsible for updating the acceleration and speed of the particles based on the error value.

- **Transmission rate R:** This is usually expressed as a number of packets per simulation step. In this work, we start the simulation program with the maximum rate $(1\,packet/step)$.

- **Flock sizes:** The flocking system can be constructed from a huge number of particles; to simplify the simulation, this system uses twenty particles $(N = 20)$ distributed in the environment.

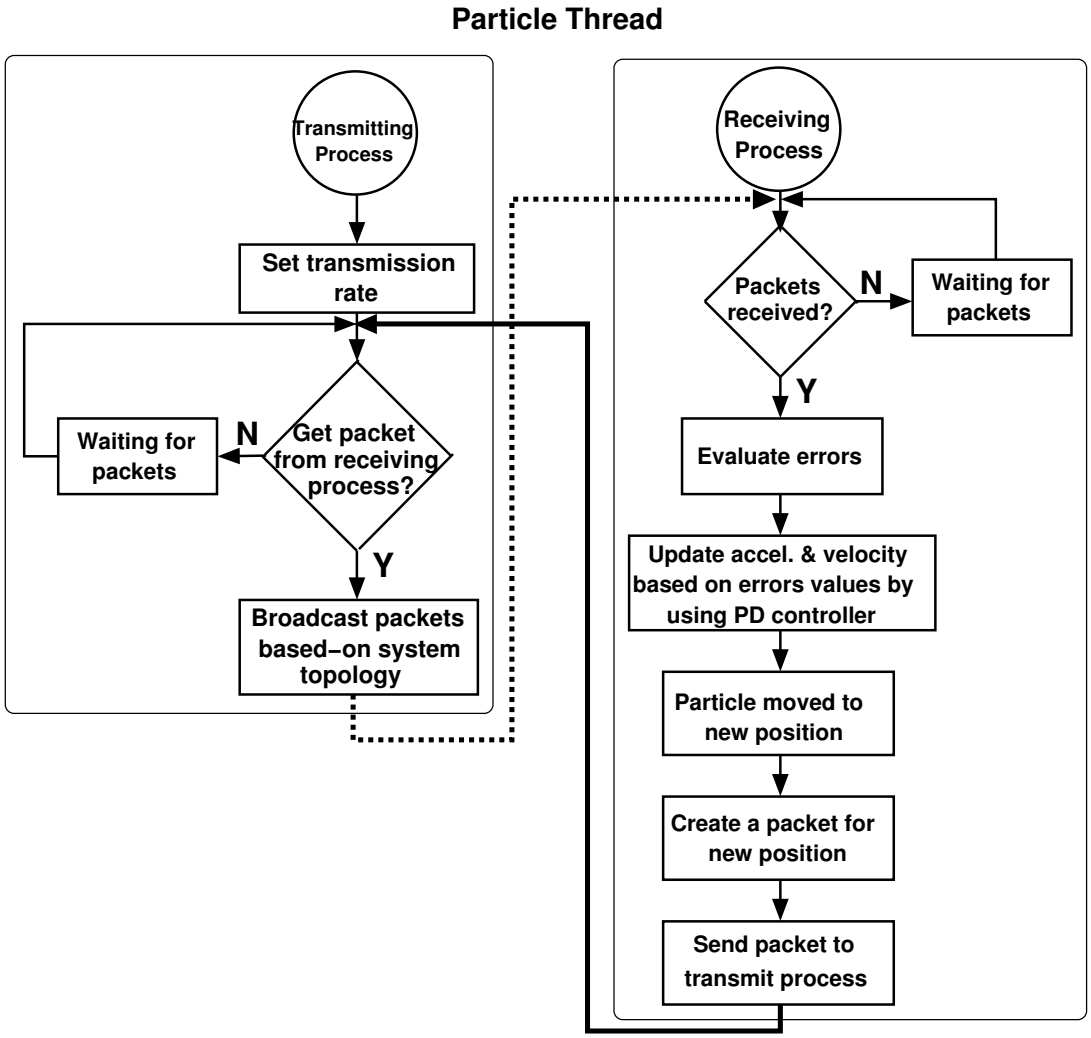


Figure 6.9.: Interaction between the transmitting and receiving process

6.4.2. Boundary Conditions

The acceleration and speed are bounded based on the error value. For example, if the distance between particle (P_1) and particle (P_2) is greater than (L_d), each particle should increase their velocity to reduce the distance up to (L_d) value. On the other hand, if the acceleration value is greater than the maximum level, it will be cut back to the maximum value, which means the speed of the particles is limited, or bounded. In this work, the whole system is optimised with acceleration range $(-0.2\,to\,0.2)$ and speed range $(-1.0\,to\,1.0)$[1].

[1]The choice of acceleration, speed ranges and micro controller parameters from a private correspondence/communication to my supervisor Fernando Rodriguez.

6.4.3. Simulation Scenarios

The equations of motion employed in this simulation are symmetrical because all the particles are identical. The simulation program affects the particles' movement by modifying only their acceleration; their velocity and position will be updated based on the acceleration value. The simulation takes place on a two-dimensional axis $(2D)$. Each particle has corresponding equations for movement on both X and Y axes. The positions, velocities and accelerations of the particles are all $2D$ vectors (x, y).

6.4.4. Evaluating Convergence Point

The convergence point of the whole system (i.e. the point at with the particles in the system have stabilized into the desired pattern around the particle leader) is evaluated based on the Cumulative Moving Average (CMA) concept, which is a type of finite impulse response filter used to analyse a set of data points by creating a series of averages of different subsets of the full data set [146, 147]. In this system it is simply used to measure the average error rate of the particles' positions until the MA is less than a specific error-tolerant value. As mentioned before, the convergence module is responsible for detecting the convergence point of the system [148]. As shown in Figure (6.10), all the particles except the leader should send the total error value of their position at each simulation step, so that as a first step, a simple moving average (SMA), which is the unweighted mean of the previous $N-1$ error values, can be calculated. Then, at each simulation step we can calculate the cumulative average (ε) and compare it to a defined error-tolerant value; the result will decide whether the simulation is to be stopped.

For each particle, the total error is determined depending on errors in x-axis and y-axis that as expressed in Equations (6.3) and (6.4).

$$dx_k = \frac{1}{L}\sum L_d - L_{d-actual} \tag{6.3}$$

$$dy_k = \frac{1}{L}\sum L_d - L_{d-actual} \tag{6.4}$$

Therefore, the total error is given as indicated in Equation (6.5).

$$d_k = \sqrt{dx_k^2 + dy_k^2}, \quad k = 1...N-1, \tag{6.5}$$

Finally, the cumulative average (ε) can be evaluated as indicated in Equation (6.6).

$$\varepsilon = \frac{1}{N-1}\sum d_k, \quad k = 1...N-1 \tag{6.6}$$

where:

d_k : RMS error.

dx_k : error in x axis.

dy_k : error in y axis.

L_d: desired distance.

$L_{d-actual}$: actual distance.

L: number of particles located in radius of perception.

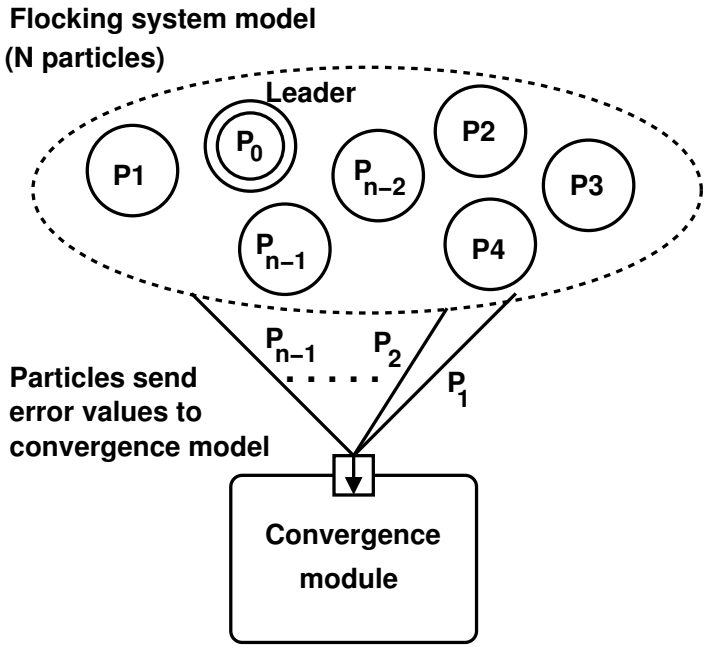


Figure 6.10.: Connecting the convergence module to the particles

6.5. Experimental Results

In this section, several simulation results are presented to illustrate the system behaviour and dynamics under different conditions, such as the system constructed as ring or fully

connected topology (Figure (6.2)), or whether the leader was fixed or assumed to be moveable. To simplify the simulation process and easy to visualise the system behaviour, we ran all experiments with a small number of particles, for example $(N = 20)$. The relative position of the particles was defined as a rectangle (with ring topology), or as mesh, when the system was constructed as fully connected (Figure (6.11)). The leader position was located in the corner or centre of the rectangle. All particles started moving with V_x and V_y based on the error value, but satisfying velocity constraints. These initial values were provided to the system by the stimulus. The interaction range between two particles ensured the relative distance (L_d) of 1 at the final destination. All the particles began moving from position $(0,0)$ and distributed themselves around the leader (if the leader was fixed) in a uniform shape based on the initial values of their relative positions. If the leader was moveable, the particles followed the leader. All these changes were made in order to validate the developed methodology and then prove that incorporating wireless communication at an early stage of the design flow is very advantageous. Multiple simulations were run to optimise the model parameters, including simulation steps, transmission rate, communication delay, speed and acceleration of the particles.

In the next section, all the experiments are carried out with the system at a high abstraction level, which means the communication between the particles is achieved by a SystemC primitive channel such as signal or FIFO. As mentioned in the beginning of this chapter, the simulation contains two parts, as follows:

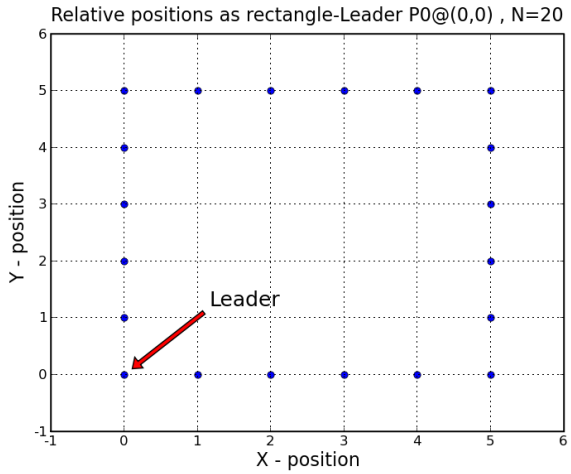
1. System modelled at high abstraction level, meaning communication is modelled based on a variable (primitive channel).

2. Refine communication by inserting our wireless communication channel, which will be covered in Chapter Seven.

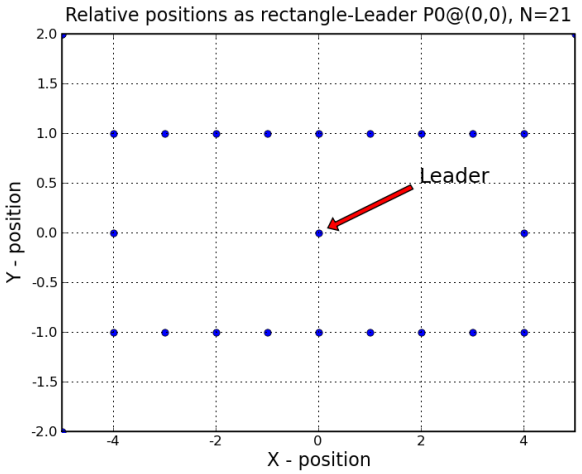
In each part of the simulation, the flocking system is constructed as ring and fully connected topologies, and both cases are investigated with leader fixed and leader movable. All the cases of system modelling that were simulated are shown in Figure (6.12).

6.5.1. System Modelling Based on Shared Variable Communication

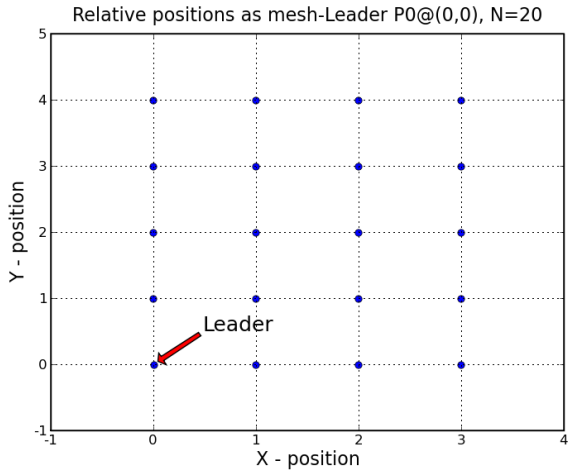
This simulation has two scenarios: in the first, the leader is fixed and all the other particles should be distributed around the leader, based on the initial values of the relative positions. In the second, the leader is movable and all the other particles should follow the leader to a specific target. Both cases will be illustrated in detail in the next sections.



(A) Rectangle with leader at the corner



(B) Rectangle with leader at the center



(C) Mesh with leader at the center

Figure 6.11.: The initial values of the relative positions

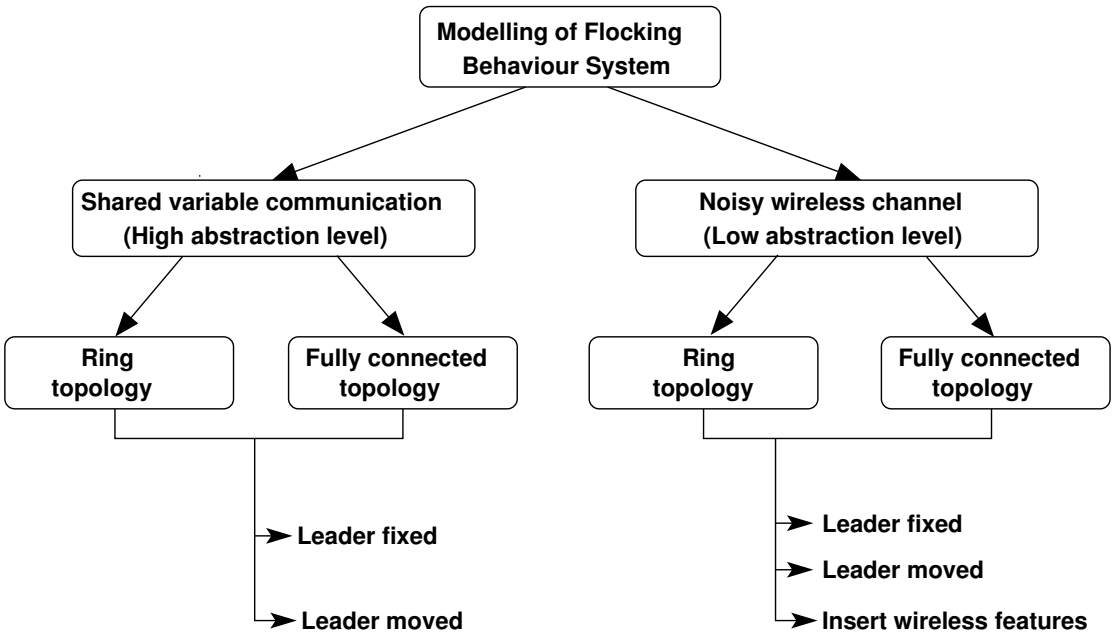


Figure 6.12.: Modelling classification of flocking behaviour system

1- First Scenario: Ring Topology

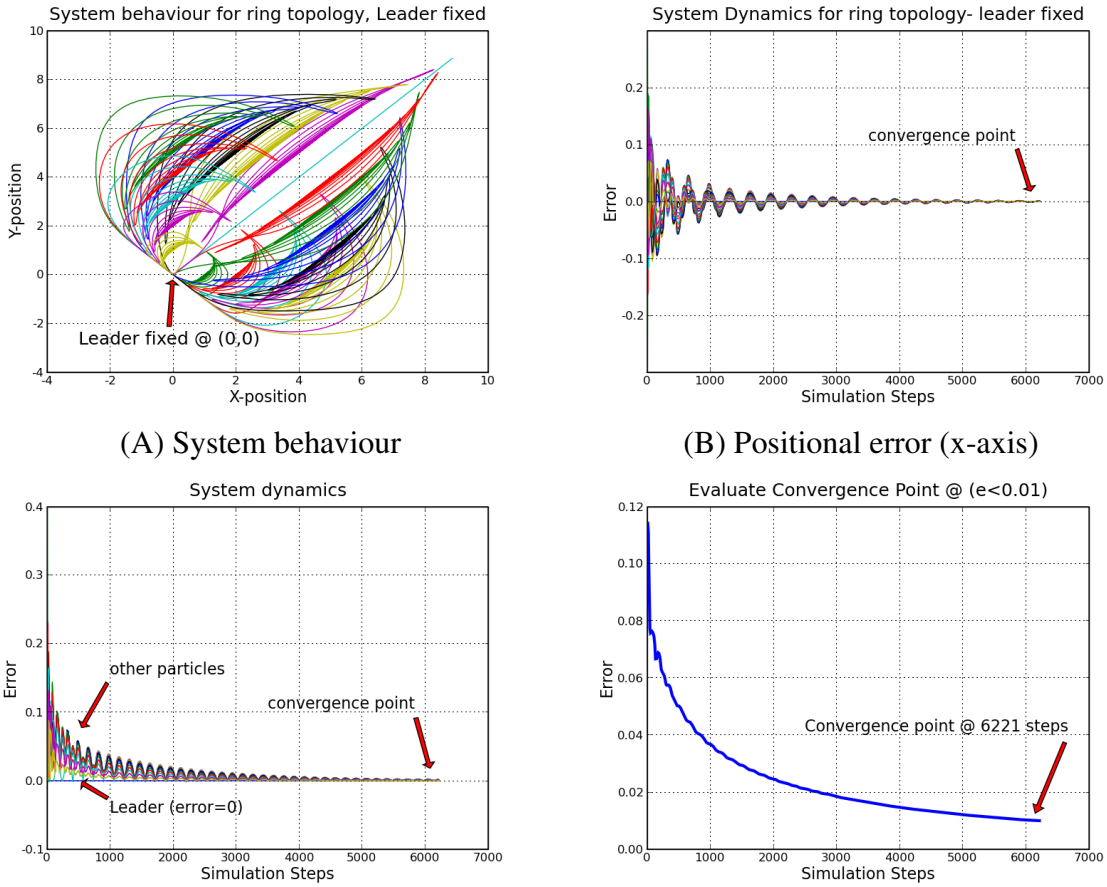
i- Leader Fixed

In the first part of this experiment, the leader is fixed at position (0,0) and 19 particles begin at the initial position (0,0). These particles navigate under the proposed control scheme based on relative position values defined as shown in Figure (6.11-A), and the system parameters indicated in Table (6.1). The communication sets in this simulation are based on ring topology, as illustrated in Equation (6.7). The particles aim to converge to the desired particles' relative positions defined at the beginning of the simulation. At the end of the simulation, the system converged and the simulation was stopped when the total error ratio was less than 0.01. Figure (6.13-A) shows the system behaviour, where each particle is represented by a different color and it clearly proves that the particles remained in the same position structure (rectangle) throughout the simulations. Thus it is proven that particles consistently and effectively avoid contact with one another. Figures (6.13-B, C) illustrate the positional error curves and the RMS error respectively, while Figure (6.13-D) illustrates the convergence point of the system, which was obtained at 6,221 simulation steps. The final position of the particles is illustrated in Figure (6.14).

$$P_0 = \{P_{19}, P_1\},\ P_1 = \{P_0, P_2\}\ \ldots.\ P_{19} = \{P_{18}, P_0\} \tag{6.7}$$

Parameters	Values
No. of particles (N)	20
Simulation time	Convergence Point (steps)
Transmission rate	1 Packets/simulation step
Acceleration range	-0.2 to 0.2
Speed range	-1.0 to 1.0
Leader	Particle P_0
Relative position shape	Rectangle, Mesh
Error (ε)	0.01
Proportional gain (K_P)	0.03
Derivative gain (K_d)	0.1
desired distance (L_d)	1

Table 6.1.: System Parameters



(A) System behaviour

(B) Positional error (x-axis)

(C) RMS error

(D) Convergence point at 0.01

Figure 6.13.: System behaviour for ring topology with shared variable fixed leader and with arrangement (A) of Figure(6.11)

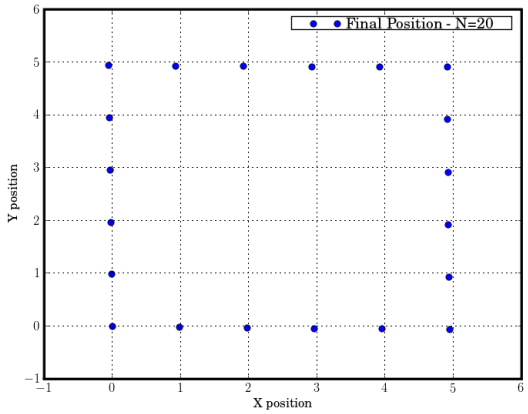


Figure 6.14.: The final positions of the particles

We will now investigate the effects of increasing relative position distance from 1 to 2. The topology is still a ring, as indicated before, but the leader position is changed to the centre of the rectangle, at position (0,0), as shown in Figure (6.11-B). Also, the number of particles involved is increased to 21 in order to create a uniform shape. The simulation is executed based on system parameters indicated in Table (6.1). At the end of the simulation, the system converged and the simulation was stopped when the error total was less than 0.01.

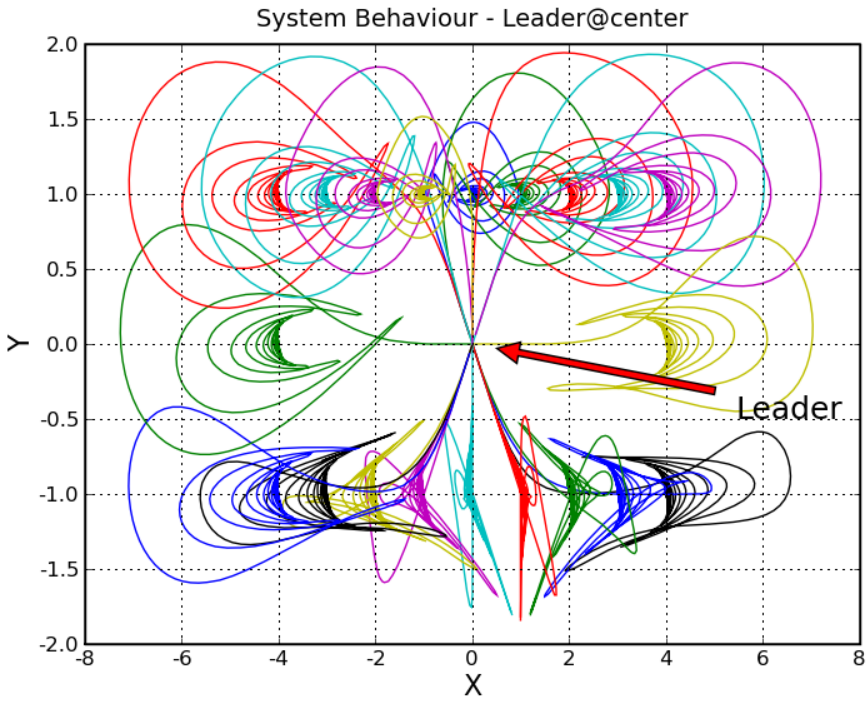


Figure 6.15.: System behaviour for ring topology with shared variable fixed leader and with arrangement (B) of figure(6.11)

ii- Leader Moved

This is the second case, where the leader is assumed to be moved and the other particles will follow the leader to the target. The leader is driven along a defined path on the x and y axes, as shown in Figure (6.16). The dashed line represents the desired trajectory that the leader should follow to reach the final target. The leader, with all the other particles, starts moving from initial position (0,0), and the relative position values that are defined as a rectangle (Figure (6.11-A)). The values of the parameters in this simulation are the same as previously indicated in Table (6.1).

In this simulation, the leader starts moving as shown in Figure (6.16) into position (20,20) in the first phase, then changes direction to position (50,-10) and finally tries to reach the target at position (180,120). Figure (6.17) depicts screen-shots A-F show the particles' trajectories on the $x-y$ plane and the achievement of flocking motion. The blue straight line in the main figure represents the leader's path, while the other lines with different colors represent the particles' paths. It also shows that the positions of all particles will finally converge to the leader's final position. Screen-shot A, (0,0) denotes the starting point of all particles, and other screen-shots (B-E) show how the particles follow the leader. In the last screen-shot, F, the leader reaches the target position (180,120) and all the particles follow it. Figures (6.18-A, B) illustrate the positional error curves and RMS error respectively, while Figure (6.18-C) illustrates the convergence point of the system, which was obtained at 44,927 simulation steps. The final position of the particles is illustrated in Figure (6.18-D).

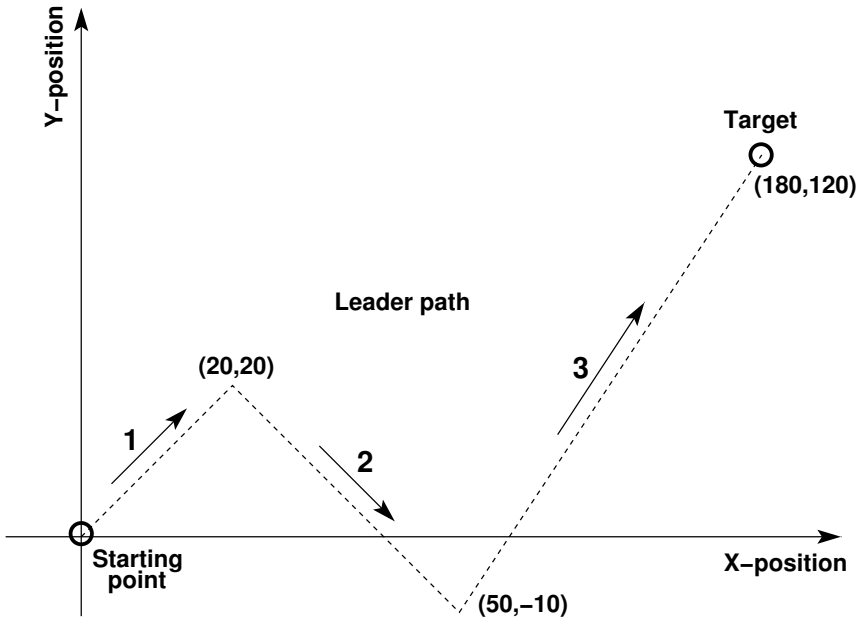


Figure 6.16.: Leader's path to the final destination

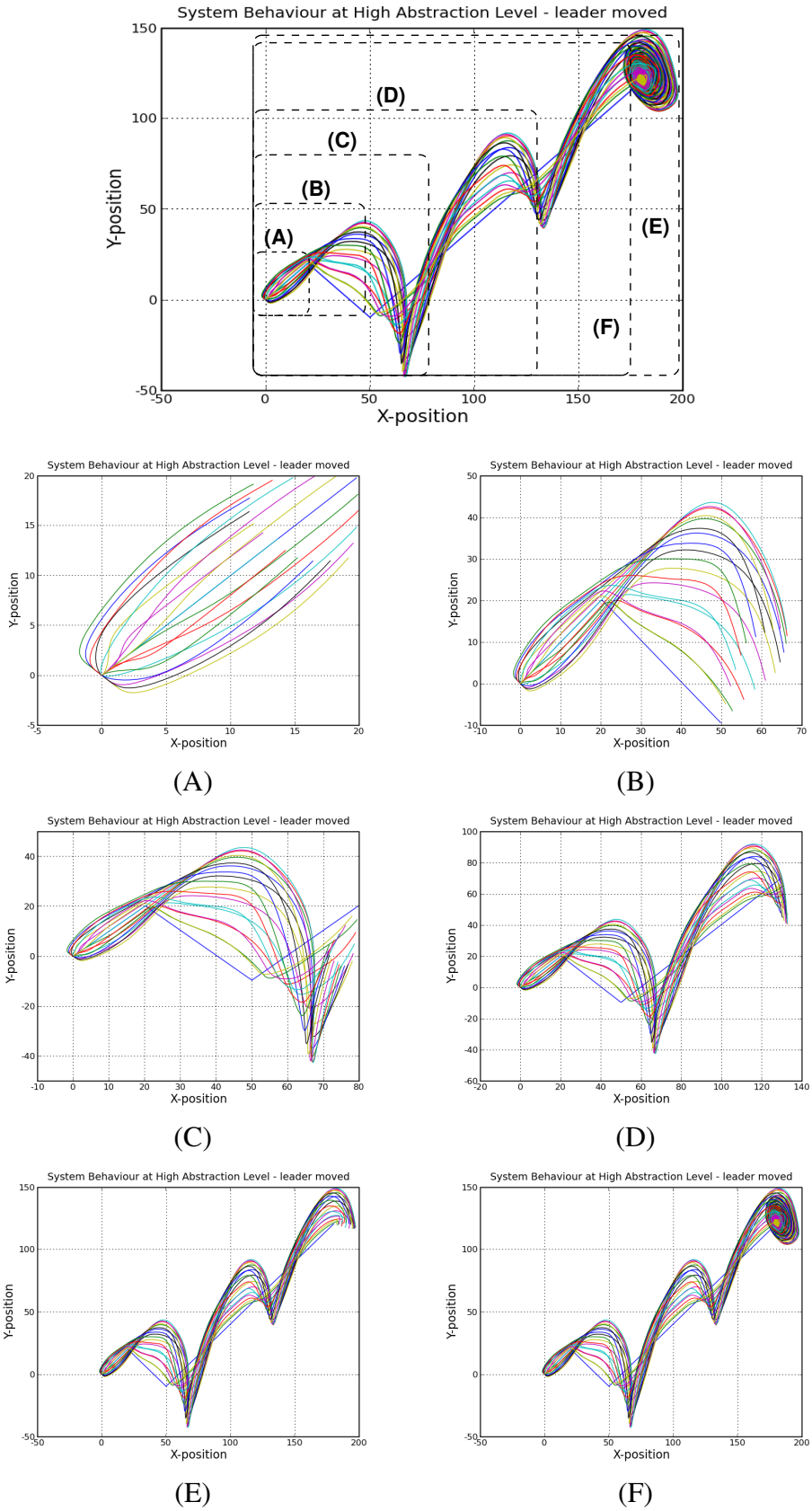
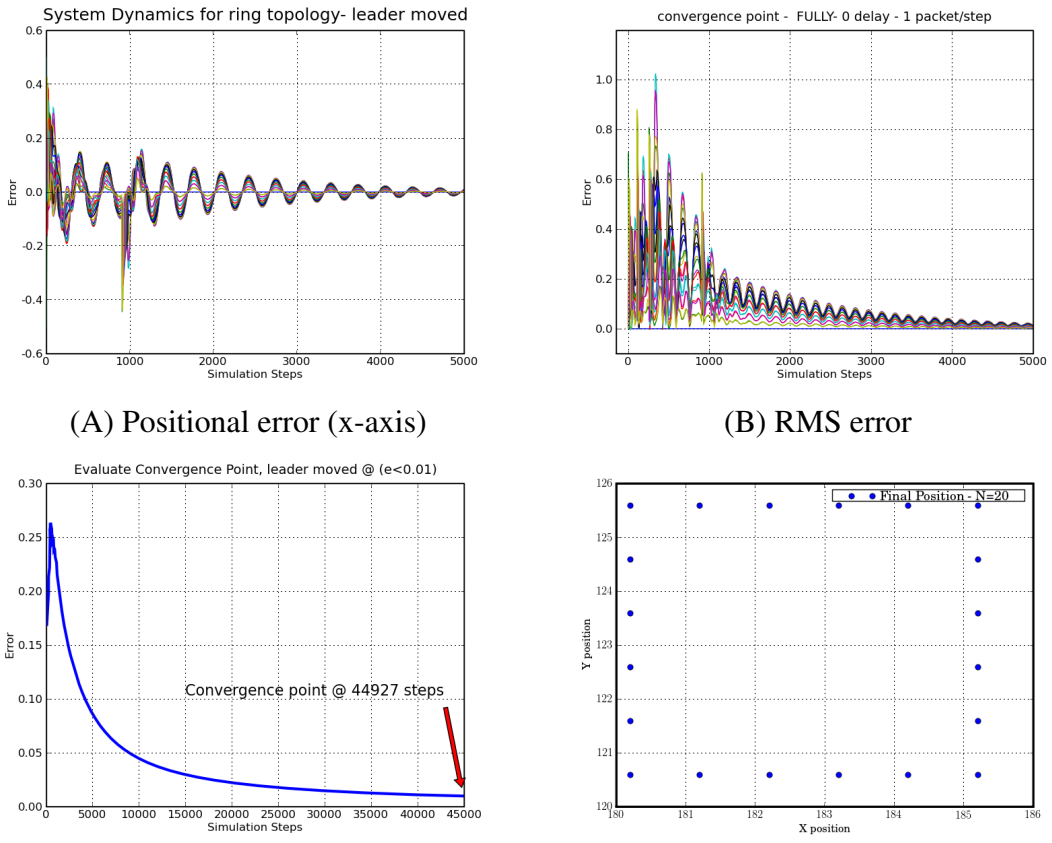


Figure 6.17.: Details of system behaviour for ring topology with shared variable moved leader with arrangement (A) of Figure(6.11)



(A) Positional error (x-axis)

(B) RMS error

(C) Convergence point at 0.01

(D) Final position of the particles

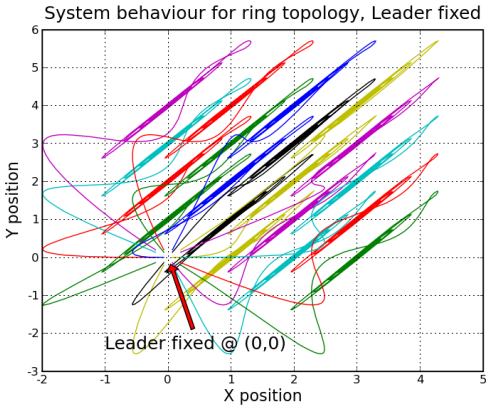
Figure 6.18.: System convergence

2- Second Scenario: Fully Connected Topology

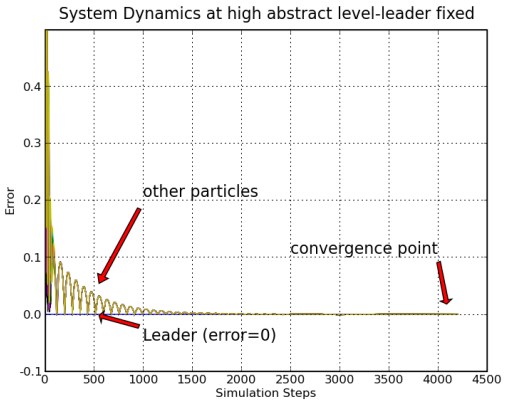
In this experiment, the relative position of the particles is distributed as mesh (Figure (6.11-C)), so each particle can communicate with all the other particles. As indicated in the first experiment, there are two cases within this scenario, as follows:

i- Leader Fixed

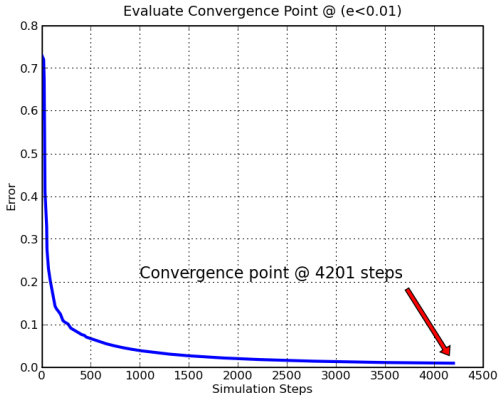
In this simulation, the leader is fixed at position (0,0) and 19 particles start from initial position (0,0). The relative positions of the particles are shown in Figure (6.11-C), which is defined as mesh. The communication sets in this simulation are based on fully connected topology, which means a broadcast mode, as illustrated in Equation (6.8). Figure (6.19-A) shows the system behaviour based on the system parameters indicated in Table (6.1). The system behaviour clearly shows that the particles reach the final positions defined in the relative positions. Thus it is proven that particles consistently and effectively avoid contact with one another. Figure (6.19-B) illustrates RMS error curves and Figure (6.19-



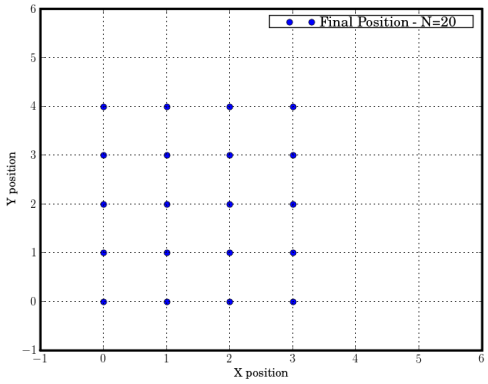
(A) System behaviour



(B) RMS error



(C) Convergence point at 0.01



(D) Final position of the particles

Figure 6.19.: Details of system behaviour for fully connected topology with shared variable fixed leader with arrangement (C) of Figure(6.11)

C) illustrates the convergence point of the system, which was obtained at 4,201 simulation steps. The final position of the particles is illustrated in Figure (6.19-D).

$$P_0 = \{P_1 ... P_{19}\},\ P_1 = \{P_0, P_2 .... P_{19}\}\ ....\ P_{19} = \{P_0 ... P_{18}\} \tag{6.8}$$

ii- Leader Moved

As previously indicated, in this second case the leader is assumed to be moveable and the other particles will follow it to the target. All of the particles move from initial position (0,0) and the relative positions values defined as shown in Figure (6.11-C). The values of the parameters in this simulation are the same as previously indicated in Table (6.1). Screen-shots A-F in Figure (6.20) show the particles' evolution in position and the achievement of flocking motion. Screen-shot A (0,0) denotes the initial position of all the particles. In the last screen-shot (Figure (6.20-F)), the leader approaches the target posi-

tion (180,120) and all the particles follow it. Figures (6.21-A, B) illustrate the positional error curves and RMS error respectively, while Figure (6.21-C) illustrates the convergence point of the system, which was obtained at 20,968 simulation steps. The final position of the particles is illustrated in Figure (6.21-D).

6.5.2. System Convergence versus Number of Particles in Terms of Different Transmission Rates

In this experiment, we need to evaluate the system convergence point against the increasing number of particles. Then we investigate effects over different transmission rates, which means the effect of adding more particles with different transmission rates. The system parameters of this experiment are illustrated in Table (6.2). The number of particles is increased from $(N = 10)$ to $(N = 500)$, while the transmission rate (R) is changed from the maximum value $(R = 1\ packet/step)$ to $(R = 0.0625\ packet/step)$. The convergence point of the system is evaluated through these two approaches; the first one based on particle position and the other based on particle energy.

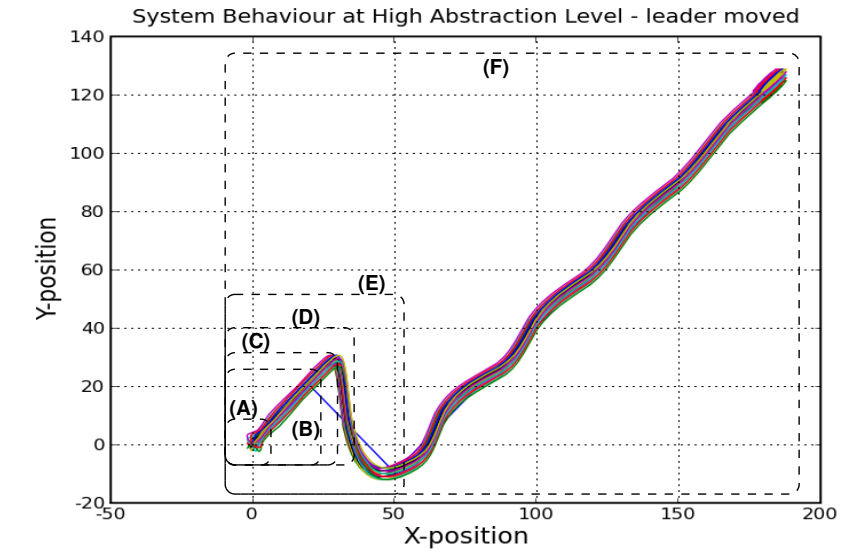
Parameters	**Values**
No. of particles	10 to 500
Transmission rate	(1 pkt/step) to (0.0625 pkt/step)
Leader	Particle P_0 - Fixed
Error (ε)	0.01
desired distance (L_d)	1
shape of relative positions	rectangle
Topology	ring

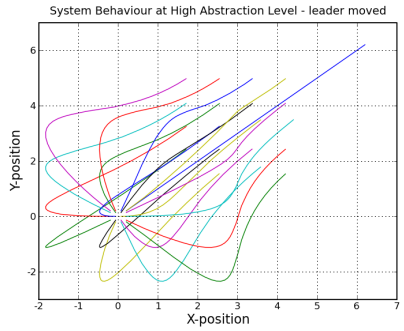
Table 6.2.: System convergence versus number of particles in terms of different transmission rates - system parameters

1- System Convergence Based on Particle Position

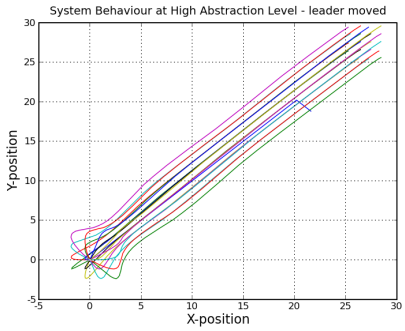
In this approach, the total error (d_k) is evaluated based on particle position, which means (dx) and (dy) represent error in particle position. Therefore, d_k is expressed as shown in Equation (6.9). Figure (6.22) illustrates system convergence versus number of particles with different transmission rates.

$$d_k = \sqrt{dx^2 + dy^2} \tag{6.9}$$

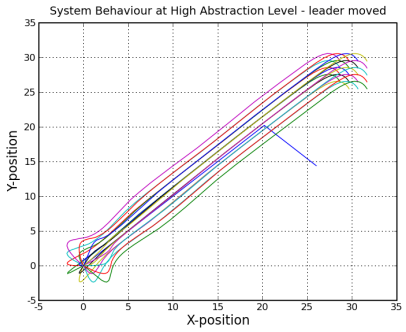




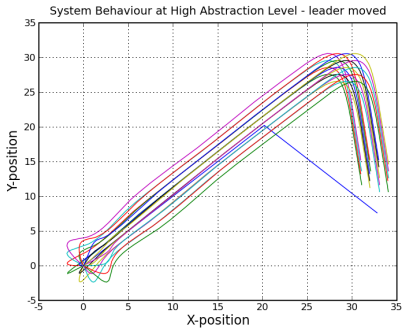
(A)



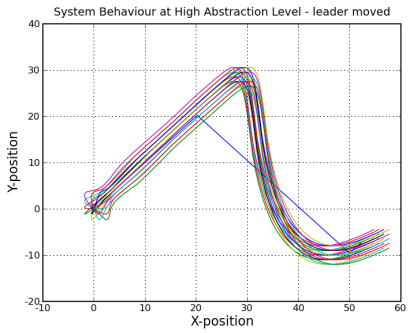
(B)



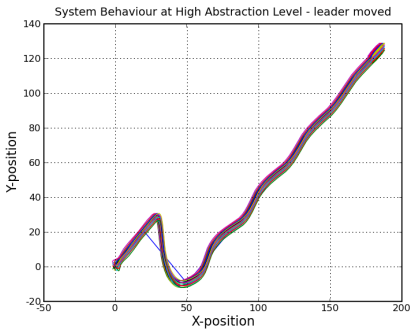
(C)



(D)

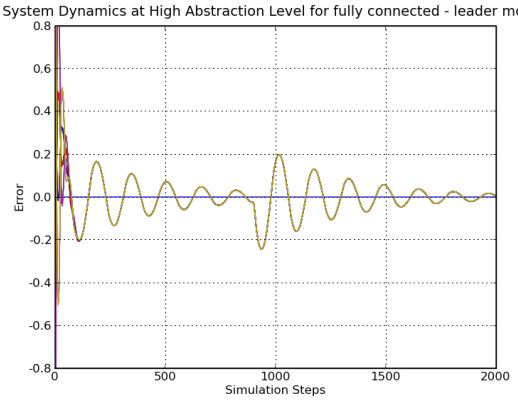


(E)

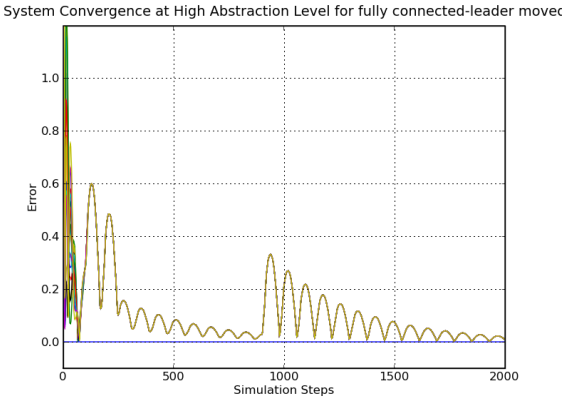


(F)

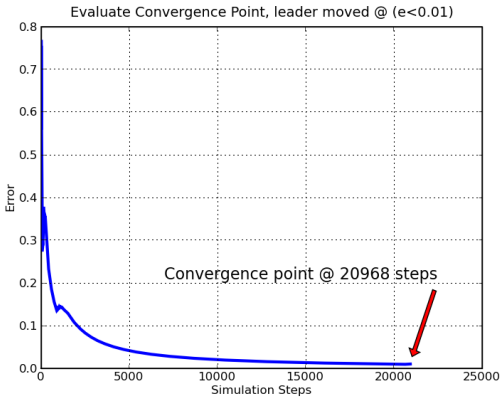
Figure 6.20.: Details of system behaviour for fully connected topology with shared variable moved leader with arrangement (C) of Figure(6.11)



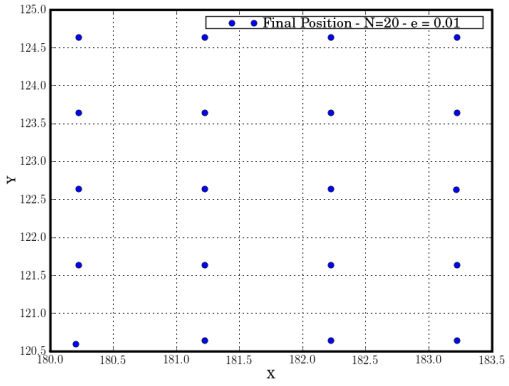
(A) Positional error (x-axis)



(B) RMS error



(C) System convergence



(D) Final position

Figure 6.21.: System convergence

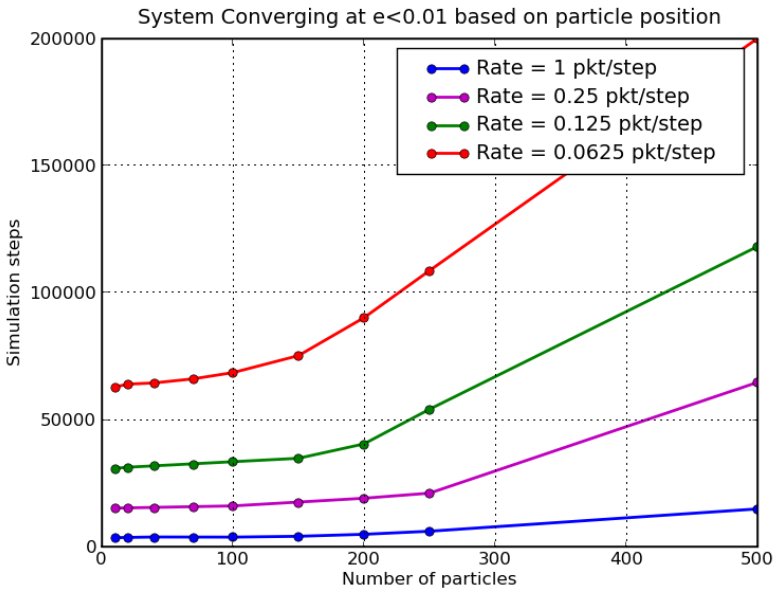


Figure 6.22.: System convergence based on particle position

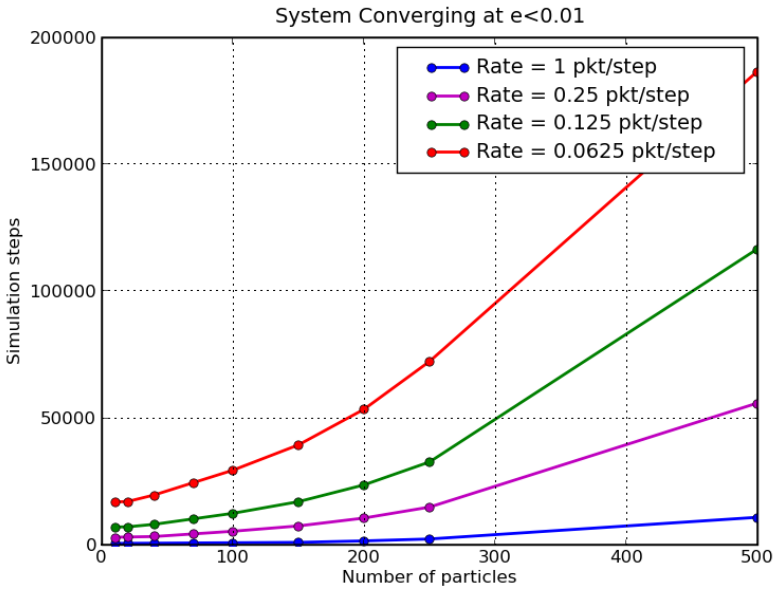


Figure 6.23.: System convergence based on particle position

2- System Convergence Based on Energy

Here, the total error is determined depending on the particles' energy, as stated in Equation (6.10), which means the total error mainly depends on the changing of the particle's energy. Figure (6.23) illustrates system convergence versus number of particles with different transmission rates. By using this approach, the system converges more quickly then the previous approach.

$$d_k = dx^2 + dy^2 \tag{6.10}$$

In this experiment, the maximum data rate is 1 $packet/step$, that we established when each particle sent a packet every simulation step. We started from the maximum data rate mentioned above and each time we reduced the transmission rate until we reach the instability point. Hence, as the data rate decreased, the system took more time to converge, because the number of packets exchanged between the particles was reduced.

6.5.3. Effect of Changing Relative Positions

In all the experiments done previously, the relative position distance (L_d) between the particles has been set to 1 in the beginning of the simulation, as shown in Figure (6.11). In this experiment, we investigate the effect of changing the relative position by increasing it

to 10, as indicated in Figure (6.24), and then checking the effect on system behaviour, as shown in Figure (6.25).

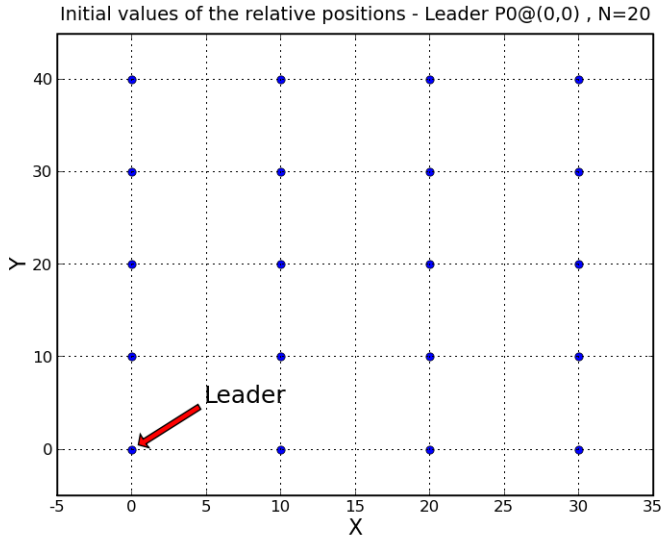


Figure 6.24.: Relative position defined with distance 10

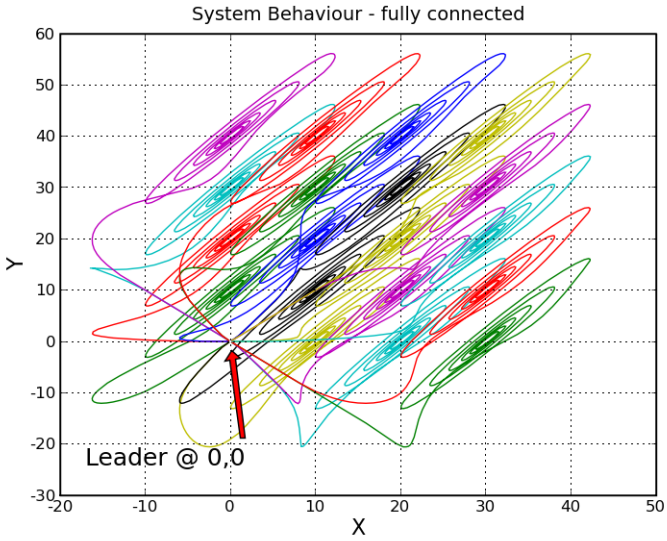


Figure 6.25.: System behaviour based on relative position distance $(L_d = 10)$

On the other hand, Figure (6.26) shows the effect of changing the relative position shape of rectangle and mesh in terms of data rates. As illustrated in Table (6.3), which summarises initial values of the system parameters, the data rate changes from the maximum value, which is 1 $step/packet$, up to 1000 $step/packet$. We note that as the data rate decreases, the system takes more time to converge.

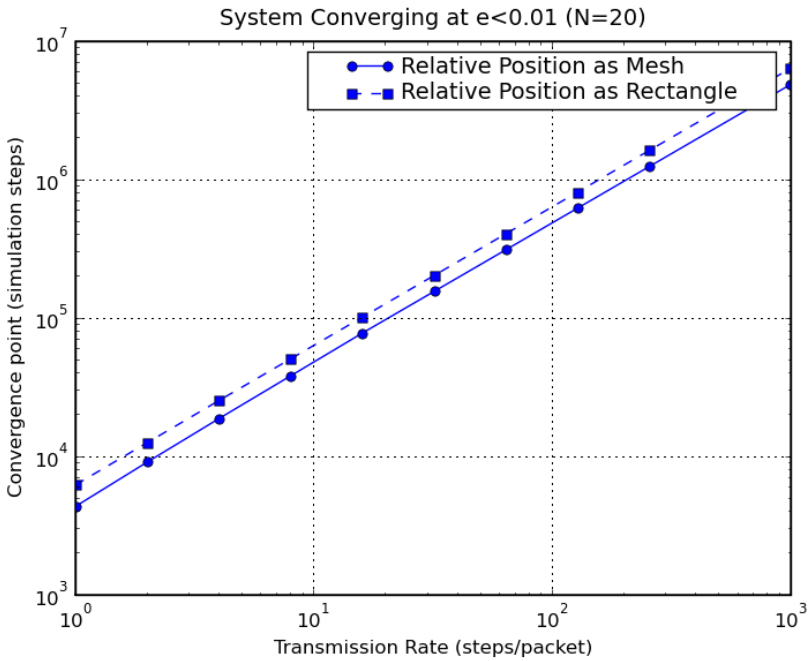


Figure 6.26.: System convergence in terms of relative positions

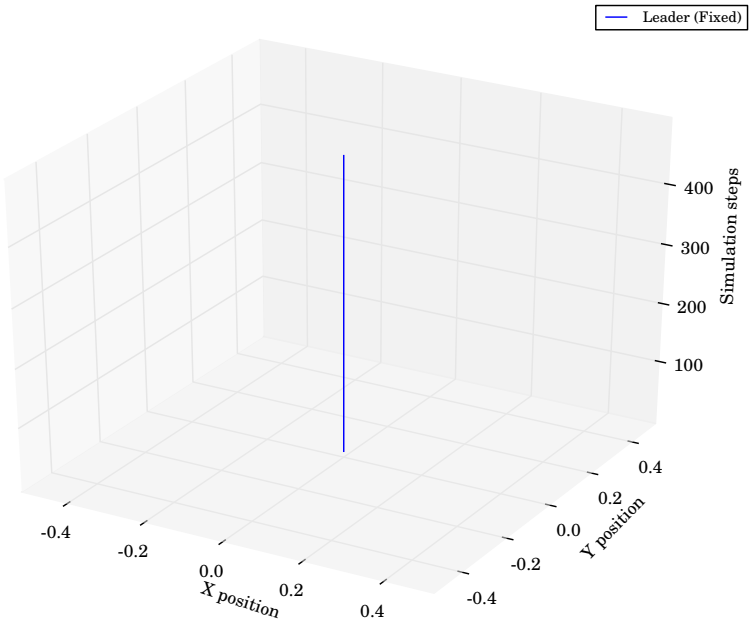
Parameters	Values
No. of particles (N)	20
Topology	Ring
Transmission rate	(1 pkt/step) to (0.001 pkt/step)
Leader	Particle P_0 - Fixed
Error (ε)	0.01
desired distance (L_d)	1

Table 6.3.: Effect of changing relative positions - system parameters

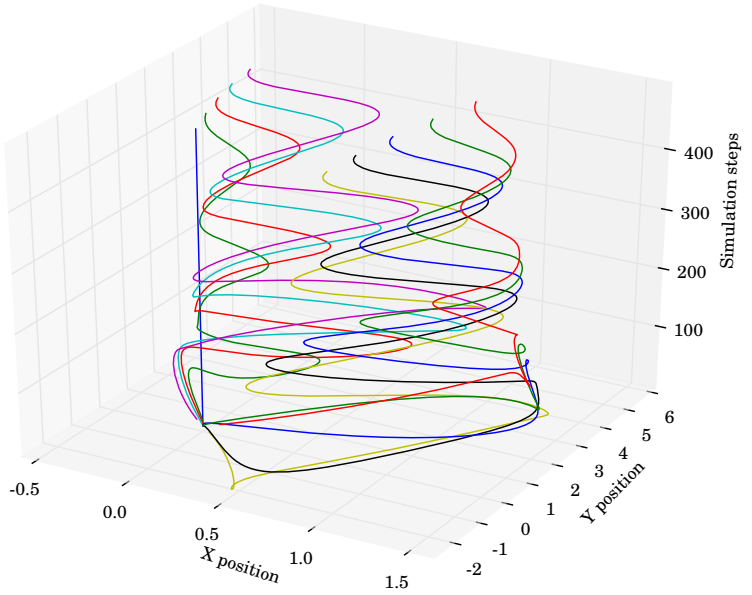
6.5.4. System Behaviour in 3D

The results shown above are presented in 2D. In this section, time is added as a third dimension to present the system behaviour in three dimensions. Consider the flocking system modelled above, with one leader and nine particles. As stated before, the initial positions of the particles are set to position (0,0), which represents the starting point. The simulation is executed based on system parameters indicated in Table (6.1), except that the number of particles is reduced to ten. In this experiment, the flocking system is constructed as ring topology and is investigated with two cases: leader fixed and leader movable. The system converged and the simulation was stopped when the error total (ε) was less than 0.01. In the first case (leader fixed), Figure (6.27-A) shows the leader fixed

at position (0,0), while Figure (6.27-B) illustrates how the other particles converged to the leader based on their relative positions.



(A) Leader fixed at (0,0)

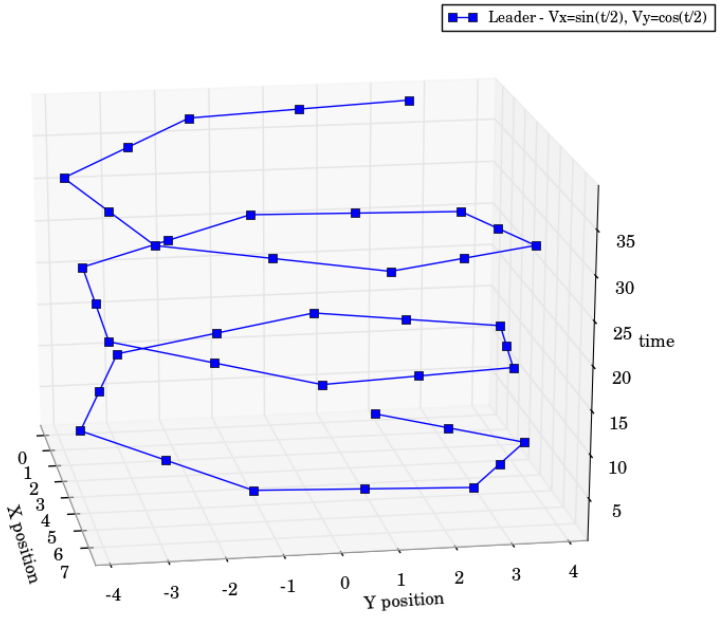


(B) Particles moving around leader

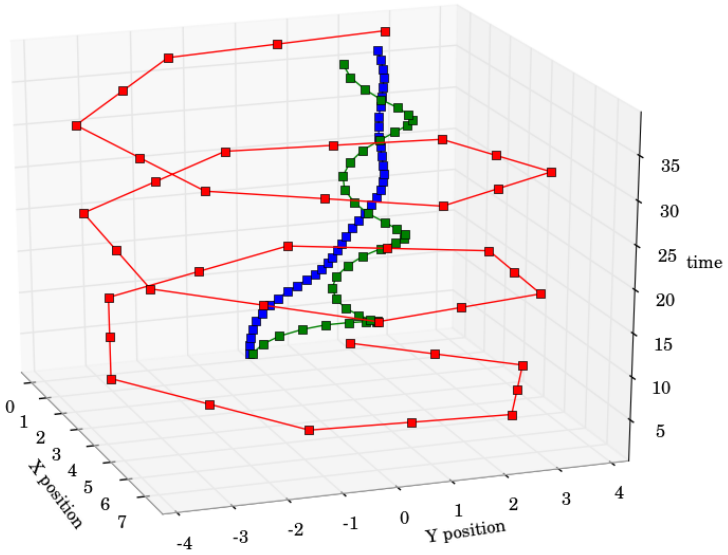
Figure 6.27.: Snapshots of flocking behaviour in 3D with the leader fixed

In the second case (leader moveable), Figures (6.28-A, B) show a flock moving in a 3D environment with a sinusoid wave profile ($sin(\frac{t}{2})$, $cos(\frac{t}{2})$) that represents the leader's path (Figure (6.28-A)). All particles' initial positions are located at original point (0,0). It is clear that all flock members eventually move together to follow the sinusoid wave of the environment (leader path), like a school of fish along ocean currents. A snapshot of two

particles following the leader is illustrated in Figure (6.28-B). Again, at the end of the simulation, the system converged and the simulation was stopped when the total error ratio (ε) was less than 0.01.



(A) Leader moving $(sin(\frac{t}{2}),\ V_y = sin(\frac{t}{2}))$



(B) Particles follow leader.

Figure 6.28.: Snapshots of flocking behaviour in 3D with the leader moving

6.6. Summary

In this chapter, the developed methodology is applied to the model of a flocking behaviour system selected as a case study, in order to validate it. The main point of modelling such system was to state that it is advantageous to have these designs that we are putting forward together since the early stages of functionality. The wireless communication of this system was created using P2P, which means there is a channel between every two particles. The communication was modelled at high abstraction level using a shared variable. To our knowledge, this is the first time that the modelling of a flocking behaviour system has been undertaken in SystemC and incorporated into a uniform design methodology, suitable for developing new technologies following the SoC design methodology. The final results of the modelled system have been validated and it has been proven that communication has a big impact in system dynamics, i.e., small changes in the wireless specifications create big changes in the system dynamics.

7. Inserting a Noisy Communication Channel

The system modelled in the previous chapter is not a practical situation, because the communication is modelled at a high abstraction level and thus cannot be used to incorporate wireless features or investigate the effects of those features. Therefore, in this chapter we extend the system to a more general case, in which wireless features such as noise and communication delay are considered, because the descriptions of such wireless features are very important aspects of channel modelling. Moreover, it is very important to introduce wireless features early on, because they will have a major effect on the system stability and system performance. Hence, as a first stage, we must insert a wireless channel model; then we can investigate the effects of introducing wireless features into the modelled system.

7.1. Inserting a Wireless Communication Channel

In order to optimise system stability in terms of communication and investigate system performance under real conditions, the communication aspect of this modelled system is refined from a high abstraction level (shared variable communication) to a low abstraction level through the insertion of our wireless channel model [133]. Then the stability and reliability of the system are investigated in order to attain the best performance under different wireless communication effects. The communication refinement process is carried out based on our developed wireless methodology described in Chapter Three. Figure (7.1) illustrates the system diagram after inserting the wireless channel. The main advantage of inserting a wireless channel model into the system is that it allows us to simulate a more realistic system when inserting noise through the channel. Moreover, it is possible to determine the effect of communication latency and communication bandwidth while still maintaining system stability. The key point is to demonstrate that one can successfully use the developed SystemC methodology to model a complex wireless communication system and to show the impacts of communication on system stability and system performance.

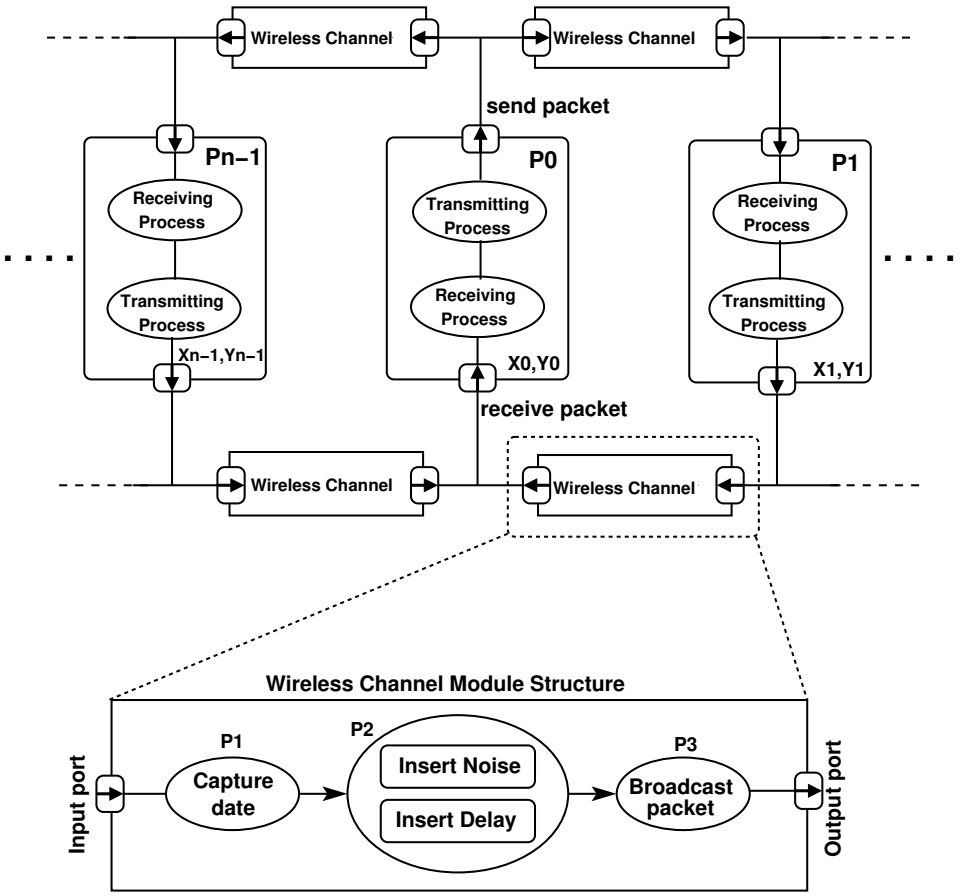


Figure 7.1.: Inserting a noisy communication channel model

7.2. Incorporating Wireless Features into the System

In the previous chapter, all the experiments were carried out with an ideal wireless communication channel model, using shared variable communication that is used in the system without the insertion of delay and noise, hence it does not affect anything and simply behaves like a wire. In a real wireless communication system, noise and delay are inherent. Therefore, inserting these wireless features is a very desirable task, because we need to simulate the modelled system with real elements, such as noise and delay, which affect system performance.

7.2.1. Impact of Noise

In communication systems, input signals as well as unwanted noise and interference are random in nature, and they are modelled by random processes [72, 88]. Impulse noise is

a non-stationary stochastic electromagnetic interference which consists of random occurrences of energy spikes with random amplitude and spectral content [72, 99]. It is critical for communication systems; therefore, the modelling of such type of noise is very important for the design and modelling of communication systems under real conditions. In this section we consider a simple digital noise model (Figure(7.2)) by modelling impulsive noise, in which the individual bits or packets are modified with a given probability (as described in Chapter Four).

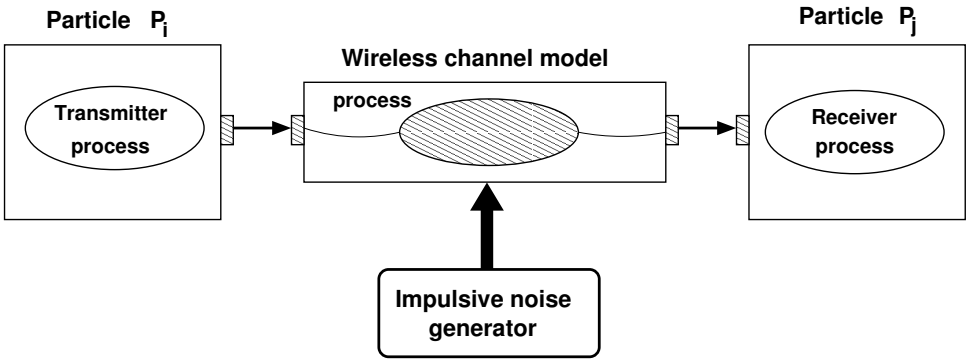


Figure 7.2.: Schematic diagram of incorporating impulsive noise into the wireless channel model

Hence, to simulate this noisy environment, a deterministic description of the channel to simulate this type of the noise is not possible; this is why the global information of the modelled system is affected by the exponential distribution function, which can be used to reflect the change of environment in a noisy environment (Figure (5.2)) [72, 99]. Meanwhile, for each particle, this information (data packet) is employed to update its state, although it has been contaminated by the noise. The method of modelling exponential distribution is based on the method used in [133]. The obtained results, as we see later in this chapter, provide sufficient conditions for the considered system to achieve flocking in a noisy environment, as well as proving the effectiveness of our wireless methodology to model complex wireless systems.

7.2.2. The Influence of Communication Delay

Delay in communication systems refers to the amount of time it takes for the packet to travel from the transmitter station to the receiver station over a communication channel [88]. To model our system in a realistic environment, we consider delay issues. Consider a time-delay (τ) that influences the system stability as well as convergence of the system. In this case, it is desirable to explore the impact of incorporating this communication delay (τ) into the system through our wireless channel model until the critical stability point

($\tau_{critical}$) is reached. We start from an ideal case, representing a zero communication delay ($\tau = 0$), and we increase delay by small increments. Hence, the modelled system will be investigated when $0 \leq \tau \leq \tau_{critical}$. At each increment, we find the system convergence point, until we reach ($\tau_{critical}$) , which is our target in this experiment.

7.3. Experimental Results

This section extends the results in Chapter Six by inserting a noisy wireless communication channel and then by incorporating wireless features such as noise and communication delay. Therefore, to verify the result of inserting a wireless communication channel, we provide some computer simulation results to illustrate the system behaviour and dynamics. The values of the parameters in this simulation are illustrated in Table (6.1); they are the same as those used in Chapter Six. For example, the system is constructed as ring or fully connected topology (Figure (6.2)), and in some cases the leader is fixed while in other cases it is assumed to be moveable. Hence, this section illustrates an extension of the results established in Chapter Six and presents the results of the system behaviour after communication has been refined by inserting a wireless channel model and then comparing the results in both cases to prove that the developed SystemC methodology has been used successfully to model wireless systems. The effects of incorporating wireless features such as communication delay and noise through the channel model are then investigated, which means there is a great impact between communication and system stability that we need to show.

7.3.1. Inserting a Wireless Channel

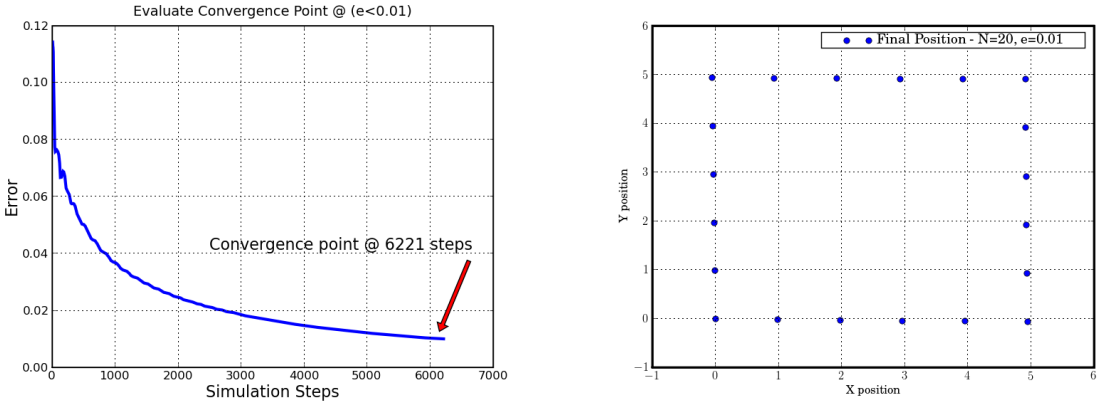
In this section, suppose we have a system of 20 particles, as in Chapter Six. The simulation has two scenarios, as mentioned before; in the first the leader is fixed and in the second the leader is moved to a specific location. As a first step, we need to incorporate the channel in the absence of communication delays and noise in order to test whether it is inserted successfully into the system. Following that, in the next section we evaluate the system convergence point in each case, after incorporating wireless features and investigating changes in the system behaviour.

1- First Scenario: Ring Topology

i- Leader Fixed

In this part of the experiment we use the initial values of the parameters used in Chapter Six, which means the leader is fixed at position (0,0) and 19 particles begin at the initial position (0,0). The particles navigate under the proposed control scheme based on relative position values defined in Figure (6.11-A) and system parameters indicated in Table (6.1).

From the experiment of simulation, the system converged and the simulation was stopped when the total error ratio was less than 0.01. The system behaviour clearly shows that the particles remained in the same position structure (rectangle) throughout the simulations. Thus it is proven that particles consistently and effectively avoid contact with one another. Figure (7.3-A) illustrates the convergence point of the system, which was obtained at 6,221 simulation steps. The final position of the particles is illustrated in Figure (7.3-B). On the other hand, when we compared the results we obtained in Chapter Six with Figure (7.3-A), we can see that it is easily verified that inserting a communication channel is successful, because both behaviours are almost the same, with the same convergence point.

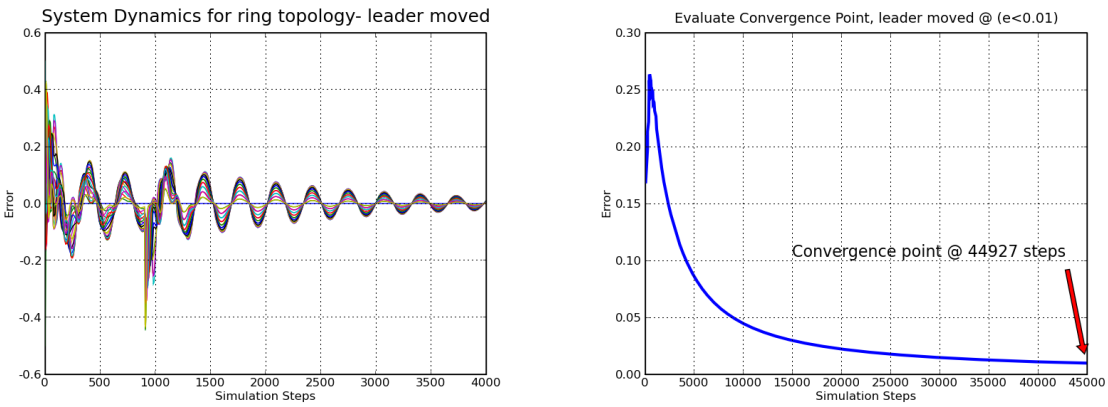


(A) System convergence (B) Final positions of the particles

Figure 7.3.: The final state of the system

ii- Leader Moved

This is the second case; the leader is assumed to be movable and the other particles will follow the leader to the target. The leader is driven by a defined path on the x and y axes, as shown in Figure (6.16). The leader and all the other particles start moving from initial position (0,0), and the relative position values are defined as a rectangle (Figure (6.11-A)). The values of the parameters in this simulation are kept the same as previously indicated



(A) Positional error (x-axis) (B) System convergence

Figure 7.4.: The final state of the system

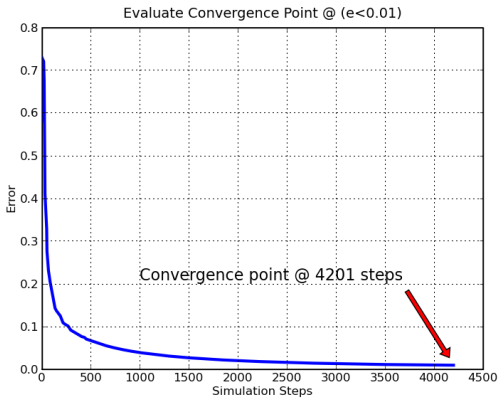
in Table (6.1). In this experiment, the system behaviour is the same that got in the Figure (6.17), which is a another prove that the wireless channel model inserted successful. Figure (7.4-A) illustrates the positional error curves and Figure (7.4-B) illustrates the convergence point of the system, which was obtained at 44,927 simulation steps.

2- Second Scenario: Fully Connected Topology

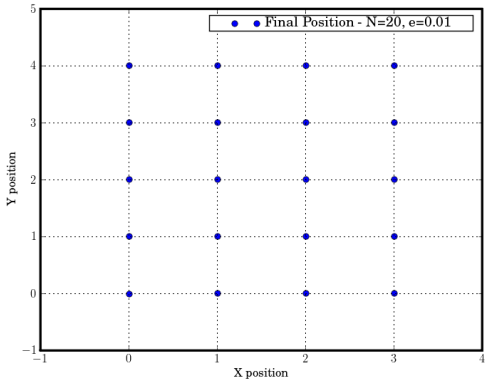
In this scenario, the main difference is that the topology is fully connected and the relative position of the particles is changed from a rectangle to a mesh, as shown in Figure (6.11-C)). Hence each particle can get information from all other particles. As indicated in the first experiment, there are two cases within this scenario, as follows:

i- Leader Fixed

This simulation involves 19 particles that try to converge to the leader's position depending on the relative position values (Figure (6.11-C)); the leader is fixed at (0,0). Again, the values of the parameters in this simulation are the same as in Chapter Six (Table (6.1)). The system behaviour, which is similar to the Figure (6.19-A), clearly showing that the particles reach the final positions defined in the relative positions. Thus it is proven that particles consistently and effectively avoid contact with one another and again the channel model incorporated successful, while Figure (7.5-A) illustrates the convergence point of the system, which was obtained at 4,201 simulation steps. The final position of the particles is illustrated in Figure (7.5-B).



(A) System convergence

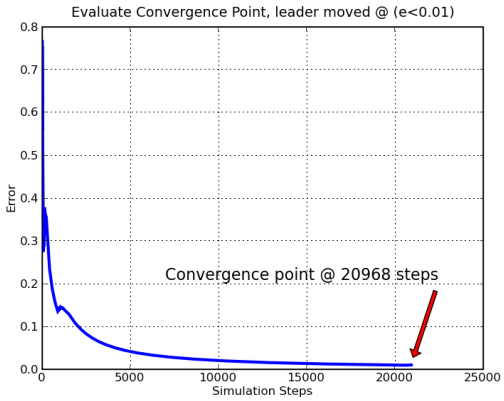


(B) The final position of the particles

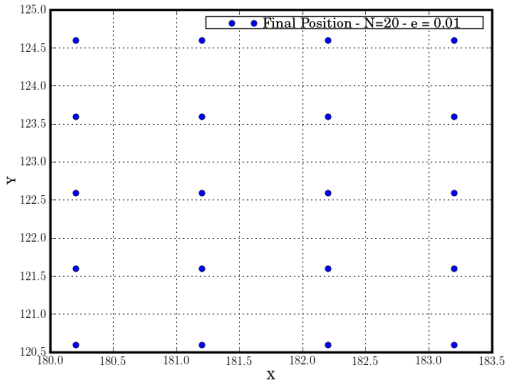
Figure 7.5.: The final state of the system

ii- Leader Moved

As mentioned in Chapter Six, it is assumed that the leader is moved to specfic location (Figure(6.16)) and the other particles will follow it to the target. All of the particles move from initial position (0,0) and the relative positions values defined in Figure (6.11-C). Again, the values of the parameters in this simulation are the same as previously indicated in Table (6.1). The system behaviour is the same that got in the Figure (6.20), which is a another prove that the wireless channel model inserted successful. Figure (7.6-A) illustrates the convergence point of the system, which was obtained at 20,968 simulation steps. The final position of the particles is illustrated in Figure (7.6-B).



(A) System convergence



(B) The final positions of the particles

Figure 7.6.: The final state of the system

At the end of this experiment, if we compare the results obtained in Chapter Six, which we use as a reference, with the corresponding results in this section (as shown in Table (7.1)), we find them to be similar, which proves that a noisy wireless channel model has been

incorporated successfully; it also proves the effectiveness of our developed methodology to model wireless systems. As a result, wireless features such as noise and delay can be inserted into the system, which we will do in the next sections.

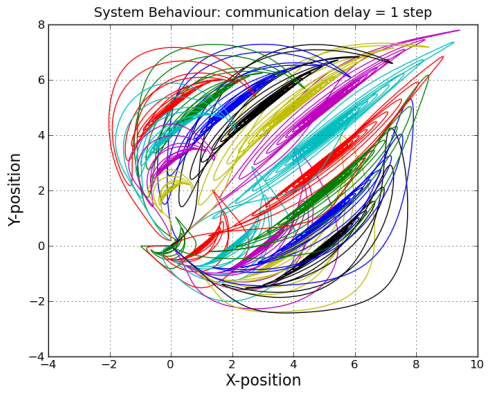
Modelling of flocking behaviour system	Modelling level	Topology	Leader mode	Convergence Point (simulation steps)
	Shared variable comm.	Ring	Fixed	6221
			Moved	44927
		Fully connected	Fixed	4201
			Moved	20968
	Noisy wireless channel	Ring	Fixed	6221
			Moved	44927
		Fully connected	Fixed	4201
			Moved	20968

Table 7.1.: System convergence points at different modelling levels

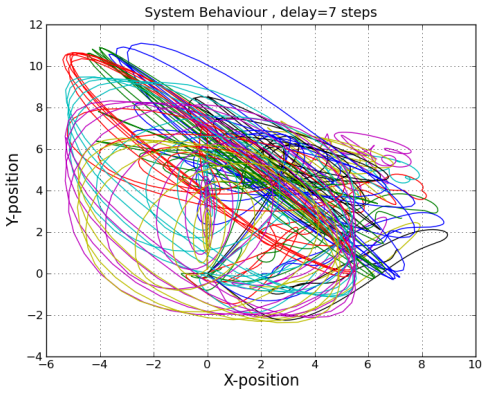
7.3.2. The Effect of Communication Delay

We present simulation results with communication delay for a system shown in Figure (6.2-A) and system parameters in Table (6.1). Figure (7.7-A, B, C, D) shows the system behaviour under the effect of communication delay $\tau = 1\ step$.

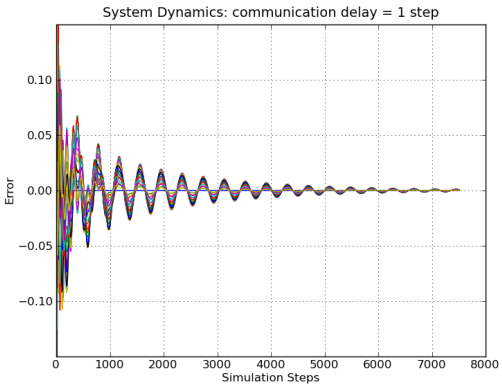
In this experiment we investigate stability of our system and we obtained $\tau_{critical} = 6\ steps$, which is the maximum delay with which the system remains stable; it is known as the critical stability point (Figure (7.8)). In this case, the communication delays do not influence the system stability; they prolong the convergence point instead.



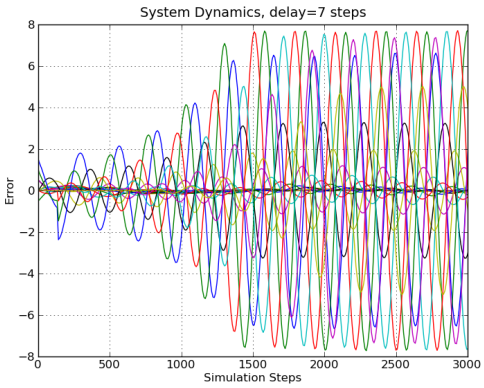
(A) System Behaviour



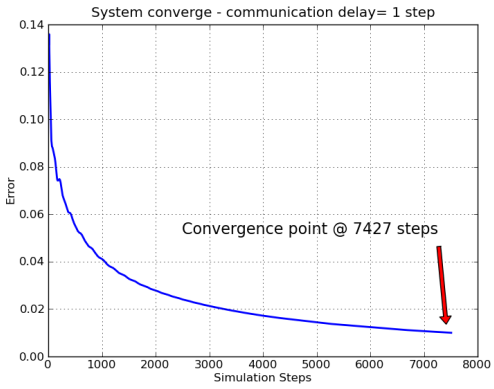
(E) System Behaviour



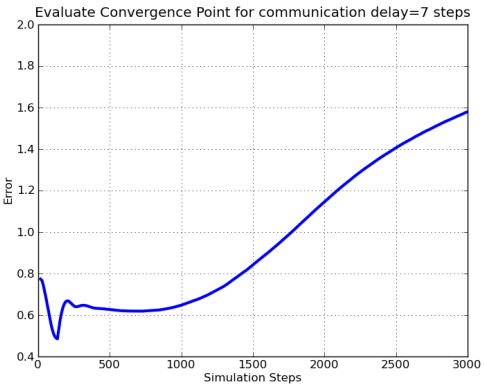
(B) Positional error



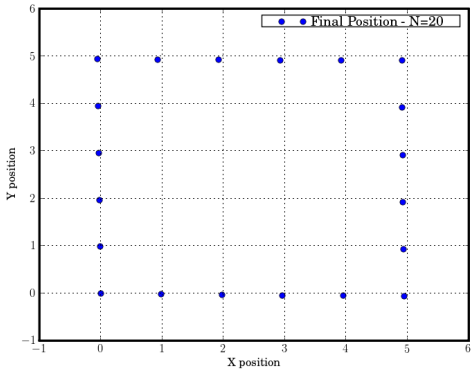
(F) Positional error



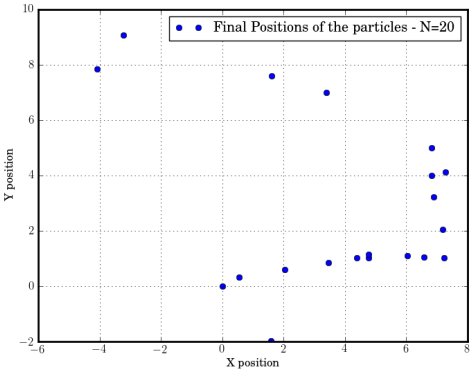
(C) System convergence



(G) System convergence



(D) Final position



(H) Final position

Figure 7.7.: System behaviour in the presence of delay, 1 step and 7 steps (instability region)

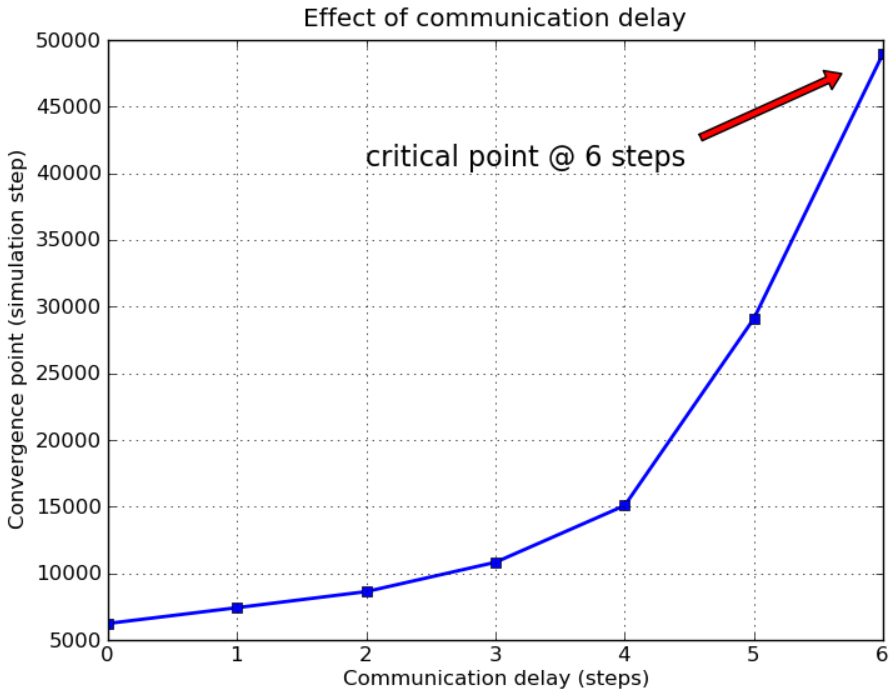


Figure 7.8.: Effect of insert communication delay

For the case where $\tau > 6\ steps$, for example $\tau = 7\ steps$, the system becomes unstable and does not converge. The system behaviour is illustrated in Figures (7.7-E, F, G, H), and Table (7.2) illustrates the convergence points of the system when $0 \leq \tau \leq 7$. The results concerning inserting a communication delay with $\tau = 7\ steps$ are interpreted according to Figure (7.9). In this figure there are three regions, which are described below:

Comm. delay (τ) *steps*	**Converging Points (simulation steps)**
0	6221 (No delay)
1	7427
2	8630
3	10830
4	15083
5	29098
6	48976
7	System unstable

Table 7.2.: Convergence points of the system in the presence of communication delay

- When the communication delay (τ) is between 0 simulation step and 6 steps, the system is stable and converges. This region is depicted by number (1) in Figure (7.9).

- When the communication delay equals $(\tau = 7\ steps)$, the system becomes unstable;

therefore ($\tau = 6\,steps$) is known as the critical stability point, represented by number (2) in Figure (7.9).

- If the communication delay is greater than 6 *steps*, as represented by number (4), the convergence point tends to infinity (Figure (7.7-G)), which means the system enters the unstable region depicted by number (3) in Figure (7.9).

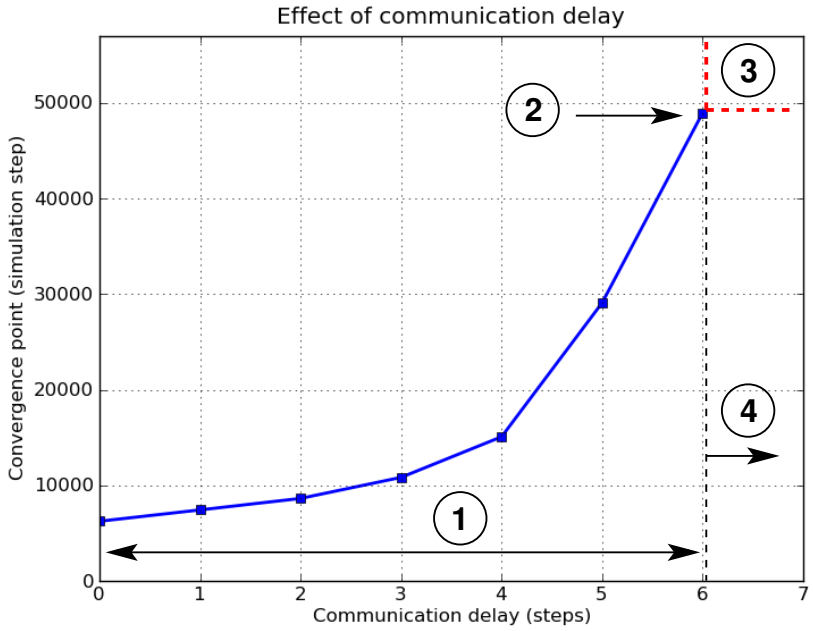
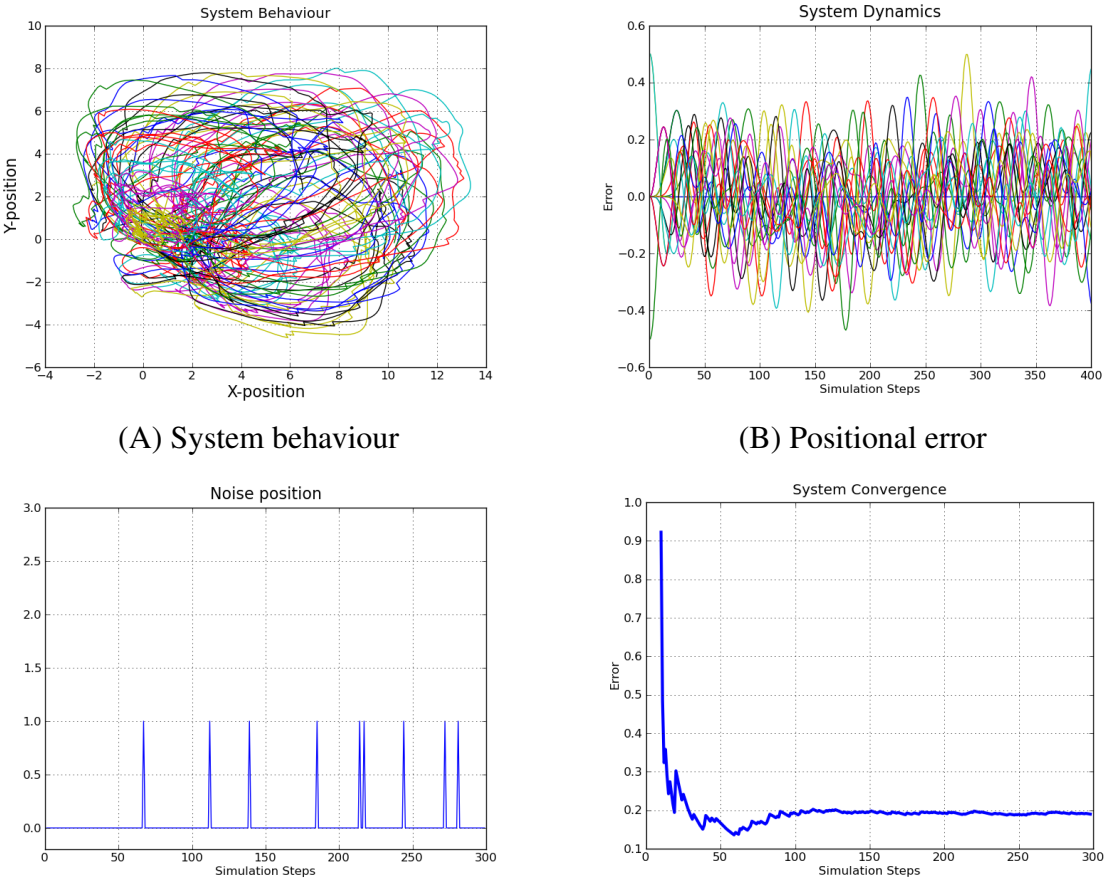


Figure 7.9.: Effect of communication delay

7.3.3. Effect of Noise

In this section we investigate modelled system performance after inserting noise, which we consider in a case where the system is influenced by bursty, impulsive noise. We consider a case of the system modelled in section 7.3.1, which is based on ring topology and where the leader is fixed. We consider independent impulsive noise as random forces depending on exponential distribution to act on individual particles. The train of noise impulses can be regarded as a renewal process, i.e. a series of random events in which the intervals between events are independent and identically distributed. Inter-arrival times between impulses exhibit statistical behaviour depending on exponential distribution, as shown in Figure (7.10-C). The effects of the impulsive noise on system behaviour are shown in Figure (7.10-A), while Figure (7.10-B) illustrates the positional error and Figure (7.10-D) illustrates the system converging, which indicates the system is not converged, i.e., the system is unstable. Hence this experiment gives us sufficient conditions for the considered system to achieve flocking in a noisy environment.



(A) System behaviour

(B) Positional error

(C) Locations of noise insertion

(D) System convergence

Figure 7.10.: The effects of inserting implusive noise

7.3.4. Investigation of Critical Unstable Point in Terms of Particle Acceleration

This experiment explores the impact of changing particles' acceleration on the system stability. We continue to investigate the system modelled in Chapter Six, section 6.5.1 (Figures (6.13) and (6.14)), with the case of ring topology and where the leader is fixed. Again the particles navigate under the proposed control scheme based on relative position values defined in Figure (6.11-A) and system parameters indicated in Table (6.1). In this system, if the acceleration value is out of range, it will be cut back to the maximum value, which means the speed of the particles is limited, or bounded.

Algorithm 7.1 Changing acceleration according to error

```
xacceleration=dx*accel+(dx-olddx)*kd;
if (xacceleration>0.2)
     xacceleration=0.2;
if (xacceleration<-0.2)
     xacceleration=-0.2;
yacceleration=dy*accel+(dy-olddy)*kd;
if (yacceleration>0.2)
     yacceleration=0.2;
if (yacceleration<-0.2)
     yacceleration=-0.2;
```

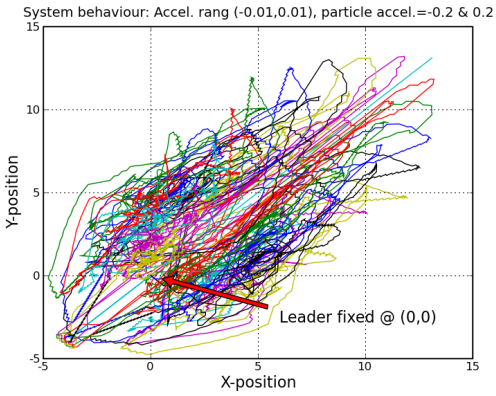
In this work, the whole system is optimised with acceleration range $(-0.2\ to\ 0.2)$ and acceleration is limited to ± 0.2, as illustrated in program segment (7.1) and system behaviour in Figure (6.13). We start by decreasing the acceleration range and setting the acceleration of the particles outwith this range as the new maximum (± 0.2) as shown in Table (7.3). Figures (7.11-A, B, C, D) illustrate the system behaviour under the effect of decreasing the acceleration range to $[-0.01, 0.01]$; the figures indicate that the system is still stable, while the instability point is reached when the acceleration range is $[-0.008, 0.008]$, as illustrated in Figures (7.11-E, F, G, H) and Figure (7.12).

Acceleration range $(square/step^2)$	Particles acceleration $(square/step^2)$	Convergence point $(simulation\ steps)$
[-0.2,0.2]	± 0.2	6221
[-0.1,0.1]	± 0.2	6513
[-0.05,0.05]	± 0.2	6975
[-0.04,0.04]	± 0.2	7018
[-0.03,0.03]	± 0.2	9034
[-0.02,0.02]	± 0.2	10099
[-0.01,0.01]	± 0.2	14127
[-0.009,0.009]	± 0.2	51268
[-0.008,0.008]	± 0.2	*system unstable*

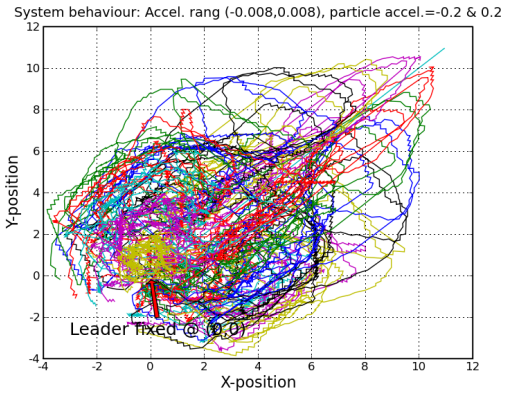
Table 7.3.: Effect of changing acceleration in system convergence

7.4. Summary

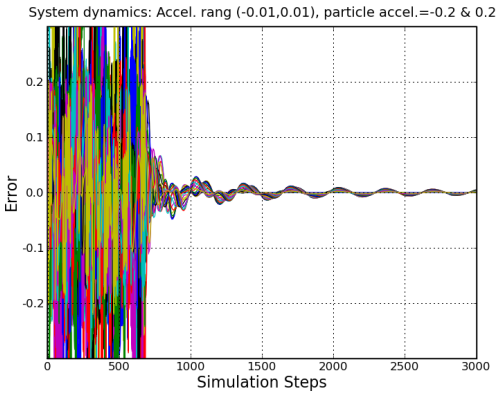
The work in this chapter is based on the work begun in Chapter Six by considering approaches to model communication for this system. The first approach considers the modelling communication at a high abstraction level (shared variable communication), which is covered in Chapter Six. The second approach consists to refine the communication by



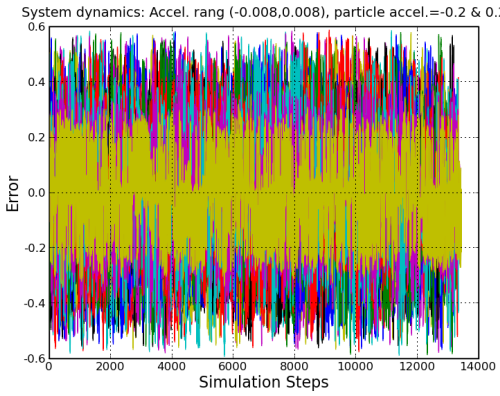
(A) System behaviour [-0.01,0.01]



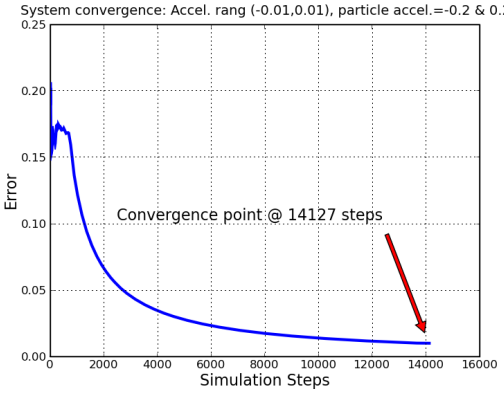
(E) System behaviour [-0.008,0.008]



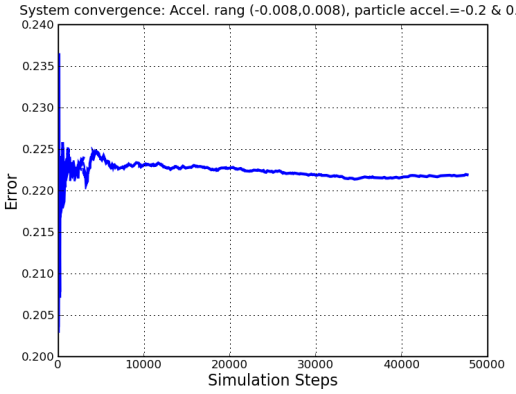
(B) Positional error, range [-0.01,0.01]



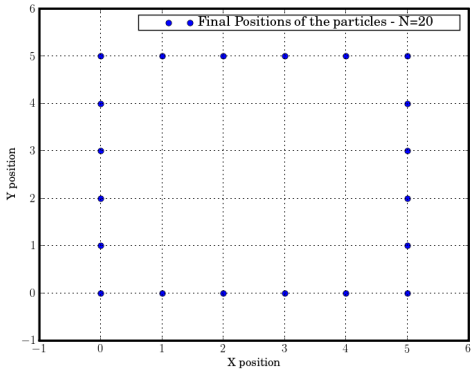
(F) Positional error, range [-0.008,0.008]



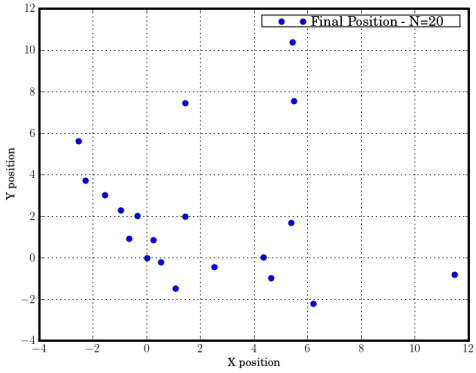
(C) System converge at 14,127 steps



(G) System not converge



(D) Final position of the particle



(H) Final position for unstable system

Figure 7.11.: Investigating instability in terms of changing particle acceleration

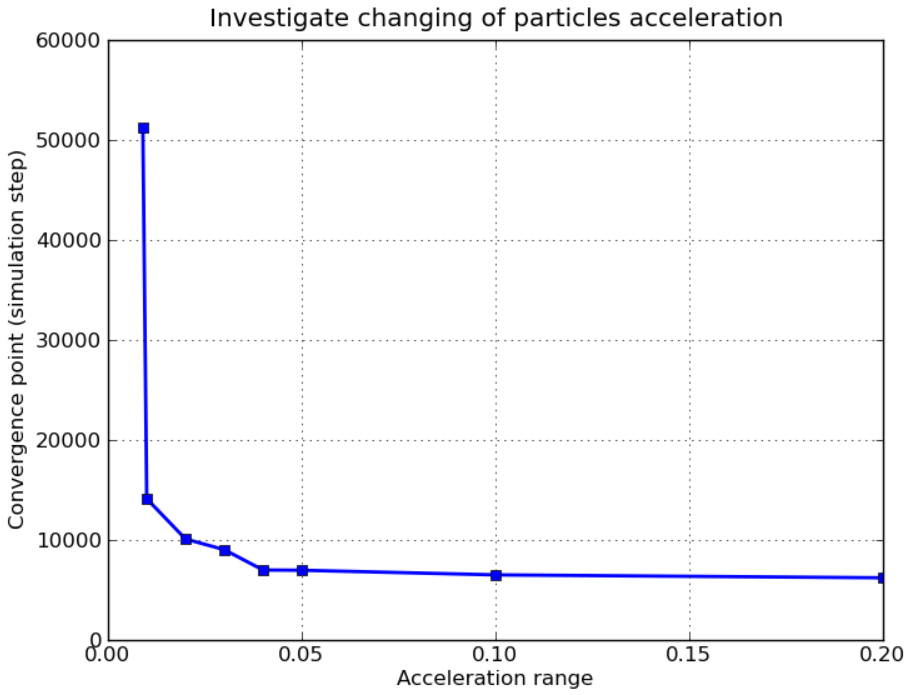


Figure 7.12.: Investigating changing of particles' acceleration against system convergence

inserting a noisy wireless channel. Wireless features such as communication delay and noise are inserted successfully and the system performance is investigated under the new conditions. The delay-free case shown of Section (7.3.1) shows the system behaviour after inserting the channel is similar to our earlier results in Chapter Six for a modelled system based on shared variables, which means the channel is incorporated successfully.

8. Measuring and Optimising Convergence and Stability in Terms of System Construction

In this chapter, the integration of communication modelling is introduced into the design modelling during the early stages of system development. We use the flocking behaviour system modelled in Chapter Six and Chapter Seven to show how the stability of the system and convergence point are measured and optimised in terms of system construction, using some important concepts of graph theory. In particular, we focus on an investigation of the system stability in terms of radius of perception.

8.1. Interaction between Control, Communication and Implementation

The flocking behaviour system modelled in this research to validate the developed SystemC methodology has a number of parameters, such as control, communication and implementation, which are interlinked and interact with each other. In this chapter, our target is to explore the interaction map of these parameters, shown in Figure (8.1). On the communication side, we need to investigate the effect of inserting a wireless channel into the system and changing the communication approaches between particles (P2P channel and shared channel). We also want to evaluate the effects of inserting noise through the channel to simulate wireless features. This will allow us to determine the effect of the communication latency and communication bandwidth (BW) to maintain system stability. On the control side, the system is investigated over different configurations; for instance, the flocking system can be modelled with the leader moved to specific position, or the leader fixed and all other particles distributed around it.

The implementation side refers to how to construct the system. This involves another level of system investigation. We need to explore the effect of changing the system topology,

for example from ring topology into bus topology, i.e all the nodes will communicate over the same communication channel, which means the communication also changes, as mentioned above. Some of these issues have been covered in Chapter Six and Chapter Seven; the rest will be covered in this chapter and the next chapter.

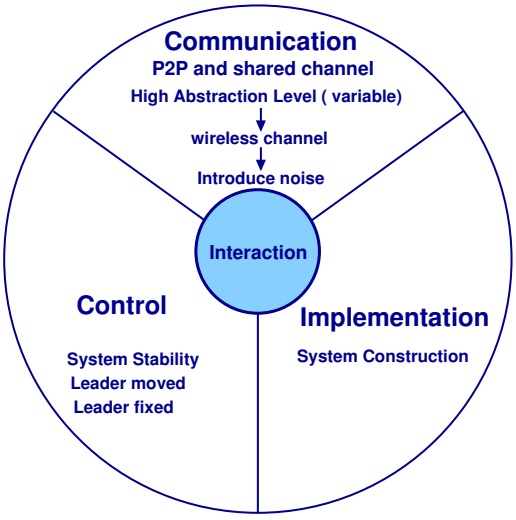


Figure 8.1.: The interaction between communication, control and implementation

8.2. Investigating System Stability in Terms of Radius of Perception

The flocking behaviour system modelled in Chapter Six and Chapter Seven is introduced to simulate the behaviour of the particles (mobile units) that form a mobile ad hoc communication network. One of the fundamental and most important issues of the ad hoc network is scalability. Scalability is the study of network stability; whenever the number of nodes and the links between these nodes changes, the topology of the network changes [149, 150]. It is meant the ability of the network to maintain its performance and efficiency as the values of some parameters of the network such as node mobility and nodal density become very large [151]. This is one of the important issues in ad hoc networks, because of the mobility of the nodes in the network. One of the fundamental questions arising during this topology change is how network performance will be affected. In this chapter we use a flocking behaviour system to show how the stability of the system and the convergence point are measured and optimised in terms of system construction, using some important concepts of graph theory [152, 153].

We begin with the basic topology (ring), and in each cycle the number of connections between particles is increased until a fully connected topology, or a fully connected graph,

is created. Thus, one of the first things this chapter does is measure the convergence and stability each time connections are added. Both converging points and stability are related to how quickly the particles assume their positions. Based on previous works [154, 155], it is apparent that graph theory concepts can play a significant role in the study and design of ad hoc networks. This work is limited to topological features of the system, i.e. we omit issues such as energy consumption. Moreover, we show how communication can have a big impact on system dynamics and should therefore be incorporated into the design process early in order to create an optimal design.

On the other hand, since a network can be modelled mathematically as a graph, therefore graph theory concepts play an important role in analysing the issues mentioned above. Graphs can be algebraically represented as matrices; hence, the study of the network can be automated through algorithms. In this chapter, to evaluate and optimise the convergence point and stability in terms of changing topology from a ring to a fully connected graph, graph theory concepts (particularly random graph theory and small world model [156, 157]) are utilised, and the matrices are employed to represent the number of connections increasing towards a fully connected graph [152, 158]. For example, consider a flock with m particles whose matrix of connection (M_c) is illustrated in Equation(8.1). This will be illustrated in greater detail later.

$$M_c = \begin{pmatrix} \times & (N_0 - N_1) & (N_0 - N_2) & \dots & (N_0 - N_{m-1}) \\ (N_1 - N_0) & \times & (N_1 - N_2) & \dots & (N_1 - N_{m-1}) \\ (N_2 - N_0) & (N_2 - N_1) & \times & \dots & (N_2 - N_{m-1}) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (N_{m-1} - N_0) & (N_{m-1} - N_1) & (N_{m-1} - N_2) & \dots & \times \end{pmatrix} \tag{8.1}$$

where:

$N_i - N_j$: Node N_i connected to Node N_j.

m : Number of nodes.

$\times$: Matrix elements not used.

According to graph theory concepts, any kind of network can be represented by a graph composed of nodes or vertices and a set of lines or links joining the nodes [152, 153]. This system uses a vertex to represent a particle and an edge to represent a link between any two particles, as shown in Figure (8.2). The set $V = \{v_k\}, k = 0, 1, ..., m-1$ includes all vertices. As mentioned above, the particles in the beginning are connected to form a ring topology. Further analysis in this chapter assumes the number of particles is too small to

simplify the graphics shapes, but a large number of particles can be used to construct a large system. To measure the convergence points in terms of different system topologies, we start with a basic topology consisting of eight particles that are connected as a ring. The number of connections between the particles increases randomly, as shown in Figure (8.3), by creating more connections, until the fully connected topology or fully connected graph is created.

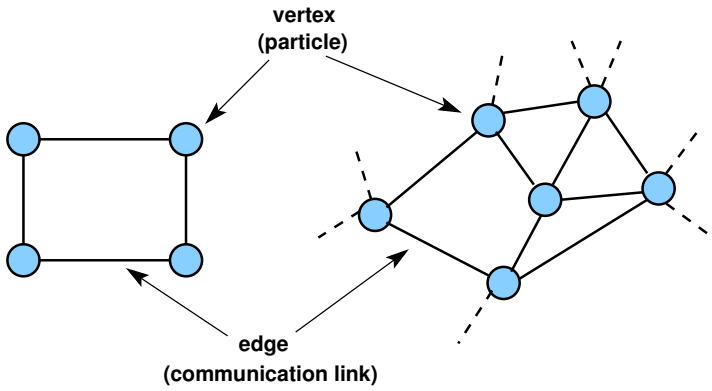


Figure 8.2.: Vertex-Edge graph components

When we evaluate and draw the RMS error of the system, a single number can be defined and measured based on total error (ε). This number represents the convergence point, which is defined in Equation (6.6). This number represents how quickly the whole system (all the particles) will converge under different conditions that will be illustrated later in the chapter. Initially, ε has a high value, but if the system is modelled successfully, the value of ε will decrease, which means the system will reach the convergence point in the next few cycles. Therefore, we can approach the convergence point, which represents how long the particles take to achieve their target. However, if the whole thing becomes unstable, the system will not converge.

8.3. Simulation Platform

We start with a ring topology, as shown in Figure (8.3). A parameter ρ is defined as the density of connections, as shown in Equation (8.2). This parameter represents the percentage of the number of random connections added to the system. Random connections are added in order to determine the value of ρ, which lies between 0 and 1. The number of edges depends on the number of particles and topology. As shown in Figure (8.3), we start from a ring topology where the number of particles equals eight and the number of connections employed to link the particles in order to create this topology is also equal

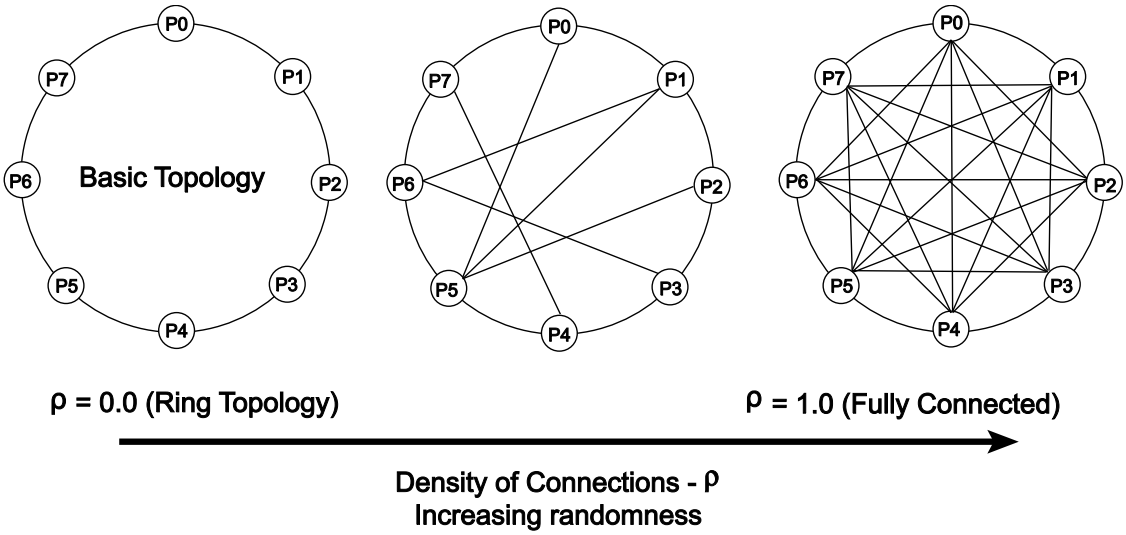


Figure 8.3.: Increasing system connections

to eight; this is equivalent to $\rho = 0$. If we randomly increase the number of connections between particles, ρ will increase. When all the particles are connected (fully connected graph), $\rho = 1$. This means that the value of ρ will change (while remaining between 0 and 1), depending on the number of links added. For this reason is called the density of connection [156, 157].

$$\rho = \frac{ec - gc}{mc - gc} \tag{8.2}$$

Where:

ρ = Density of connections and its value from 0 to 1.

ec = Existing connections = given connections+randomly added connections

gc = Given connections = N (number of connections in ring topology).

mc = Maximum number of connections = $\frac{(N*(N-1))}{2}$

In SystemC, a functional verification of the modelled system is done through simulation. This process consists of applying a stimulus to the Device Under Test (DUT) and verifying the response against an expected result. At each time step in the simulation, the processes in Figure (8.4) are applied to all particles simultaneously, and the positions and velocities of all particles at the next time step are updated accordingly.

As indicated above, this work investigates the effects of increasing the number of connections between particles against the convergence time. The number of particles is $(N = 8)$, so in a basic topology the number of connections is 8 links. Based on Equation (8.3), we

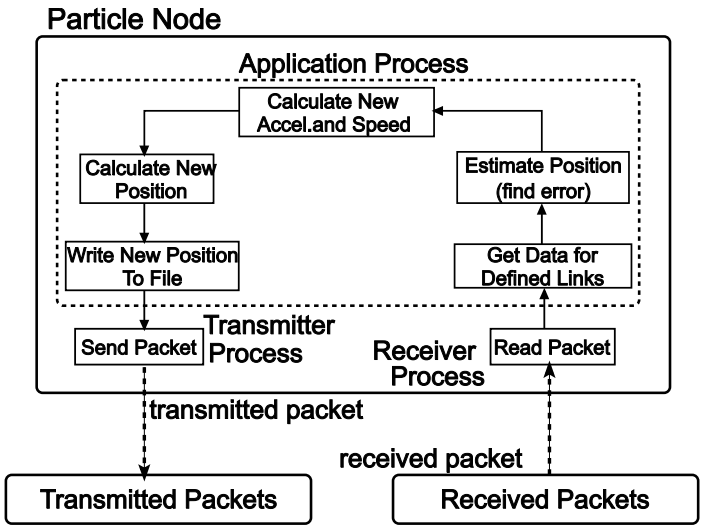


Figure 8.4.: Data Flow within the particle model

must increase the number of connections to 28 *links*, which represents a fully connected graph (maximum number of connections).

$$maxNoOfConnections = (N * (N-1))/2 \tag{8.3}$$

In order to create connection matrices that can be employed to represent the increasing number of connections, we implement a small C++ program to generate the connections randomly, but within the range based on the maximum number of connections (MaxNoOf-Connections), i.e. ranging from eight links to 28 links. The connections are provided to the SystemC program in the form of matrix (8×8), as shown in Figure (8.5), where (0) represents particle (P_i) is not connected to particle (P_j) and (1) means they are connected. Before running the simulation program, we expect the convergence time to decrease while the number of connections increases. That is to say, by increasing the number of connections, the system will reach the stability point more quickly.

8.4. Experimental Results

8.4.1. System Behaviour

The simulated platform considers eight particles $(N = 8)$ constructing a flocking behaviour system with a leader and with initial absolute positions for all the particles at $(0,0)$. The relative positions are arranged in a square shape, with the basic topology as a ring. These initial values were provided to the system by the stimulus. In this experiment, we assume

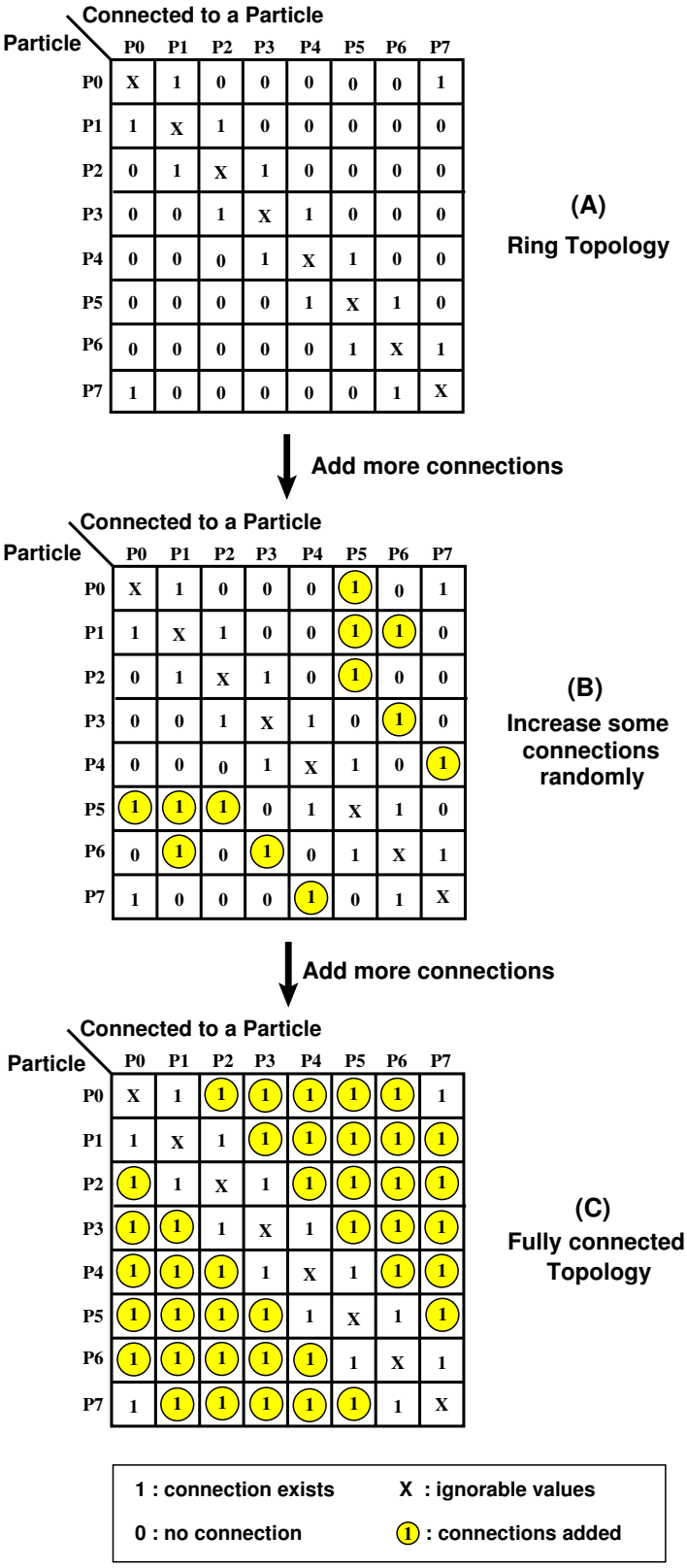


Figure 8.5.: Connection matrix

the leader is fixed and the particles begin moving from $(0,0)$, distributing themselves around the leader in a uniform shape depending on the shape created by the relative position of the particles. Multiple simulations were run to find the convergence time of the system at each level of interconnection. Figure (8.6) shows the system behaviour based on the system parameters indicated in Table (8.1). The system behaviour clearly shows that the particles remained in the same position structure (square) throughout the simulations. Thus it is proven that particles consistently and effectively avoid contact with one another. Figure (8.7-A) illustrates the changing particle velocities (for example, particles P_1, P_2 and P_3). But if the velocity value is greater than the maximum level, it will be cut back to the maximum value as shown in Figure (8.7-B).

Parameters	**Values**
No. of particles (N)	8
Error (ε)	0.01
Transmission rate	1 Packets/simulation step
Acceleration range	-0.2 to 0.2
Speed range	-1.0 to 1.0
Proportional gain (K_P)	0.03
Derivative gain (K_d)	0.1
Leader (P_0)	Fixed
Relative positions shape	square

Table 8.1.: System parameters

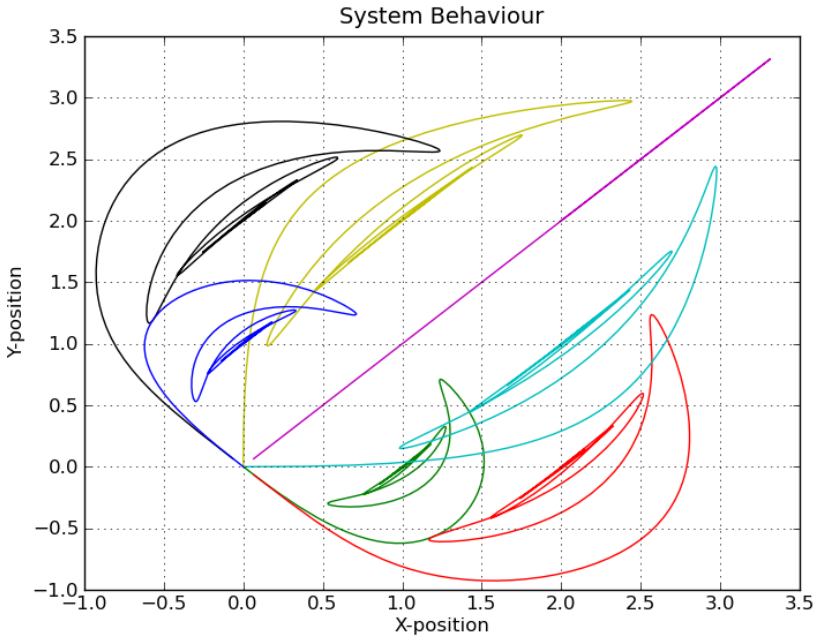
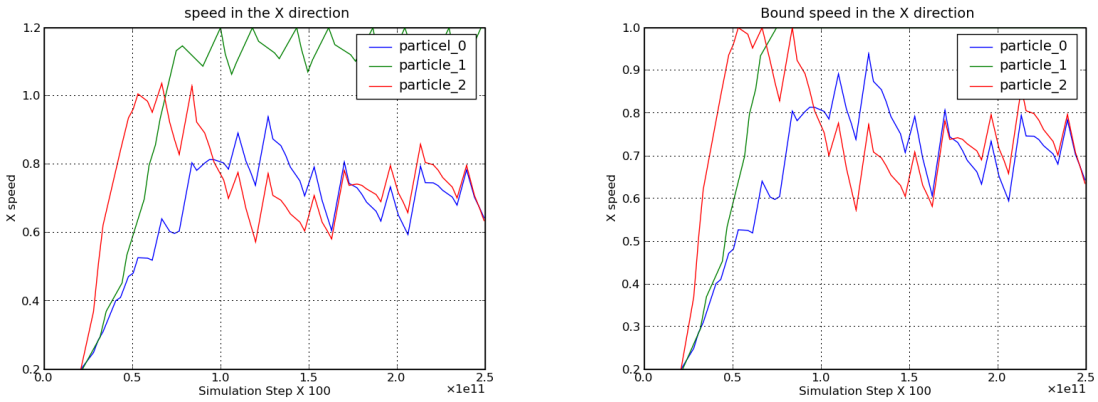


Figure 8.6.: System behaviour



(A): Unbounded (B): Bounded

Figure 8.7.: The velocity of the particle P_1 in X direction

8.4.2. System Converging Point against Number of Connections

The main aim of this experiment is to measure the convergence points and then investigate the effects of increasing the number of links between particles against the convergence points, starting with ring topology, which consists of eight particles and eight links. The number of links between particles is increased randomly by adding more links, until the fully connected topology is created. After running the simulation program, the convergence time decreases against an increase of the number of connections, as illustrated in Table (8.2) and Figure (8.8). This means that by increasing the number of connections, the system will reach the stability point more quickly, because each particle is able to get more information from the other particles.

No. of Connections	**Converging Points**	ρ
8	2530 (Ring)	0
10	2461	0.1
12	2427	0.2
14	2401	0.3
16	2384	0.4
18	2381	0.5
20	2379	0.6
22	2375	0.7
24	2371	0.8
26	2366	0.9
28	2363 (Fully Connected)	1.0

Table 8.2.: Converging time against density of connections

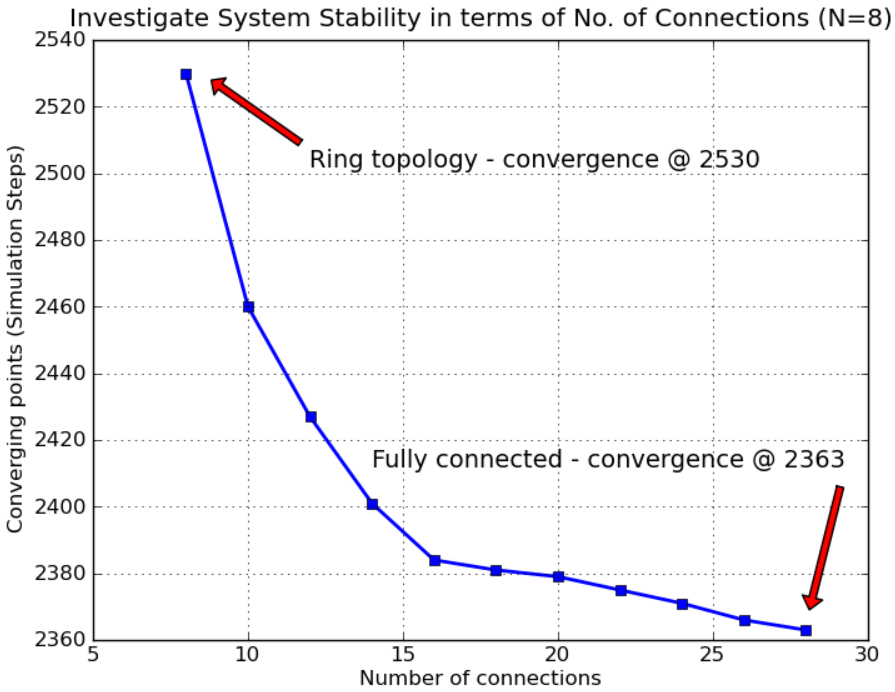


Figure 8.8.: Number of connections versus converging time

8.4.3. System Convergence Against Density of Additional Connections

Parameters	Values
No. of particles	10 to 200
Transmission speed	1 packet/step
Leader	Particle P_0 - Fixed
Error (ε)	0.01
Desired distance (L_d)	1
Shape of relative positions	mesh

Table 8.3.: System convergence against density of additional connections - system parameters

1- Evaluating Convergence Point Based on Particle Position

In this approach, the RMS error is evaluated based on particle position, as indicated in Equation (8.4).

$$d_k = \sqrt{dx^2 + dy^2} \tag{8.4}$$

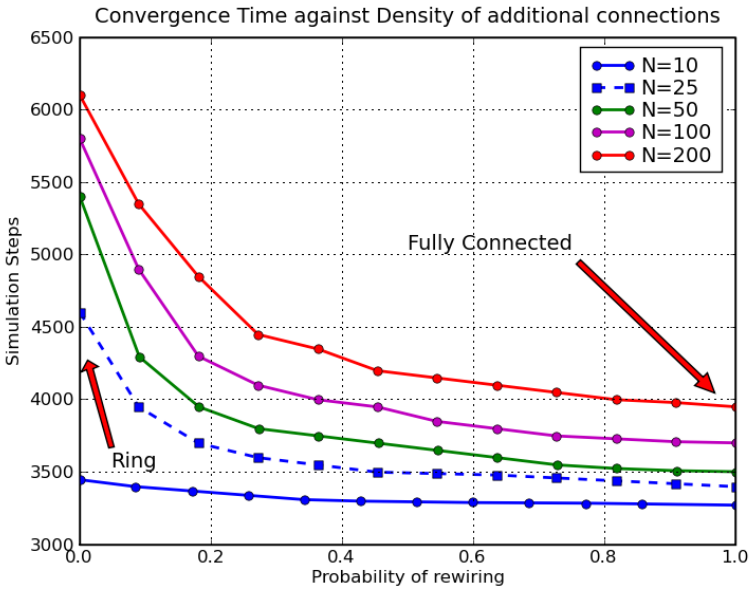


Figure 8.9.: Evaluate convergence point based on particle position

2- Evaluating Convergence Point Based on Energy

Here, the RMS error is calculated depending on particle energy, as stated in Equation (8.5).

$$d_k = dx^2 + dy^2 \tag{8.5}$$

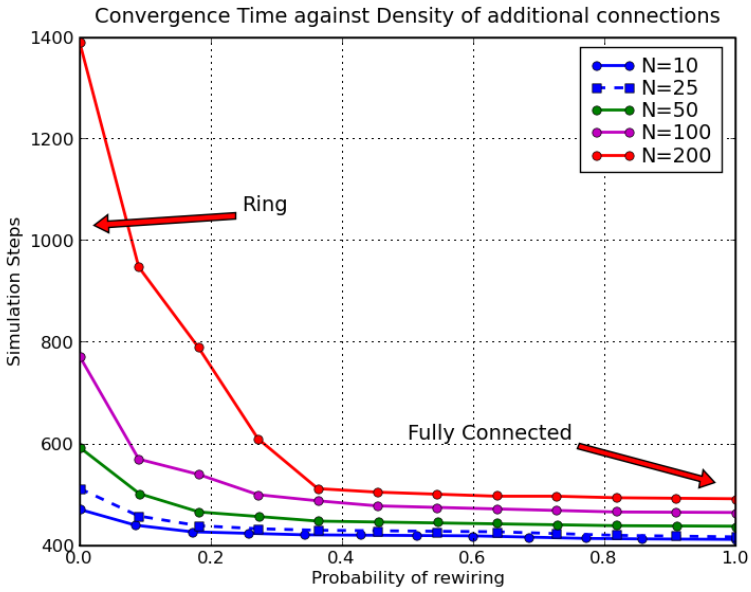


Figure 8.10.: Evaluate convergence point based on particle energy

8.5. Summary

This chapter illustrates a model for a flocking behaviour system using SystemC, and shows how the system stability is optimised using some important concepts of graph theory. Based on this work, it is apparent that graph theory concepts can play an important role in the study and design of ad hoc networks. The work here is limited to topological features of the system, i.e. issues such as energy consumption have been omitted. Moreover, this work demonstrates a simple and computationally efficient way to model a wireless communication system at system level. The model is developed at a highly abstract level, allowing for rapid simulation and early estimation, both of which are necessary for successful system development using SoC design methodology. The system has been modeled successfully; positive results representing system behaviour and system dynamics were established, which means that the developed methodology has been employed to model wireless system in an efficient manner.

9. Modelling Communication Based-on Multiple Access Protocol

The modelling of the flocking behaviour system and all its experiments described in Chapter Six, Chapter Seven and Chapter Eight are implemented based on multiple orthogonal channels, so that the wireless channel between a pair of particles can be seen as a link (P2P link), and the ignores all other transmissions and focuses only on the communication between each pair or particles. This approach is used in order to model and design a robust radio-based wireless network and to manage the complexity of the system. However, the problem with this approach is that it does not allow the modelling of the Medium Access Control (MAC) layer of any wireless protocol for the contention of the wireless medium, in the sense that it does not eliminate the channel-sharing issues that must be solved in order to maximise the aggregate data delivery capacity of the wireless system. It is a priority to allocate that capacity in a reasonably fair manner. Hence, to model and simulate a more efficient system, this chapter illustrates in detail how the flocking behaviour system can be modelled based on a shared channel, and it is better to use the new modelled system to validate the developed methodology. However, to make a transmission successful over such approach i.e, shared channel, interference and contention between the particles must be avoided or at least controlled. The channel becomes the shared resource whose allocation is critical for proper operation of the target system. Therefore, we need to used access schemes to such channels known as multiple access protocols.

9.1. Multiple Access Protocols

Multiple access protocols can be defined as approaches that allow multiple users (multiple transmitter-receiver pairs) to share a common channel [98]. The main goal of this approach is to maximise the capacity within the "local neighborhood" by trying to make every transmission count and, moreover, to allow many users to share a finite amount of

radio spectrum at the same time but without severe degradation in the system performance [89]. Because concurrent transmissions may collide and cause packet corruption, the goal is to manage which nodes are allowed to send simultaneously. These protocols do not take a network-wide view of the sharing problem, focusing instead only on "local" radio regions. They are called multiple access protocols because they arbitrate transmissions amongst multiple concurrent users [81, 89, 159].

As mentioned before, the wireless communication of this system is created through using a wireless channel model in two scenarios: the first scenario is based on multiple orthogonal channels, meaning there is a channel between every two particles – this scenario is covered in Chapters Six, Seven and Eight, and also in [19, 20, 21]; and the other is shared channel scenario, which can be used to achieve communication between all particles based on multiple access protocols; this scenario is covered in this chapter. These protocols have been designed to handle access to the shared channel and are classified into three categories as shown in Figure (9.1). In this chapter, two multiple access protocols from different categories are selected in order to validate the developed methodology over shared channel approach, because we do not need to investigate these protocols, but we need to prove our methodology under different conditions. The first protocol is Time Division Multiple Access (TDMA), which is classified as a channelisation protocol in which the available bandwidth of the channel is shared in time [22, 160]. The second protocol is Carrier Sense Multiple Access (CSMA), which is classified as a random access protocol. It was developed to minimise the chance of collision and therefore increase system performance. It uses a contention-based approach to channel access, and does not require time synchronization [23, 24]. The results of both cases are discussed and compared. Finally, we must mention that in this research, we do not need to investigate multiple access protocols, but we have to prove our methodology under different conditions, one of which is how to integrate communication with different approaches, such as P2P and shared channel to system alternative, at an early stage of the design flow.

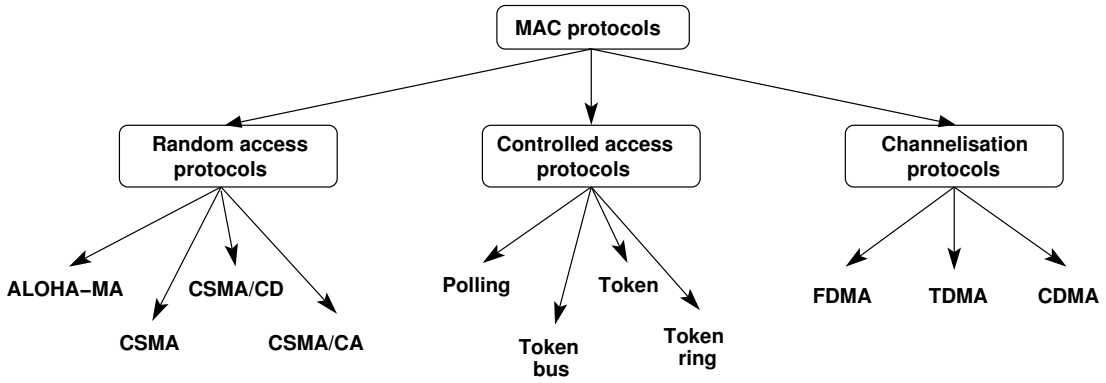


Figure 9.1.: Multiple access techniques

9.1.1. TDMA

TDMA is a channel access method for sharing wireless channels by time. It allows several users to share the same frequency channel by dividing the signal into different time slots. The users transmit in rapid succession, one after the other, each using their own time slot. This allows multiple stations to share the same transmission medium. In this approach, time is segmented into intervals called frames. Each frame is further partitioned into assignable user time slots, as shown in Figure (9.2). The frame structure repeats, so that a fixed TDMA assignment constitutes one or more slots that periodically appear during each time frame. Each station transmits its data in bursts, timed so as to arrive at the shared channel in its designated time slot(s) [24]. When the bursts are received by the shared channel, they are retransmitted on the receiving station, together with the bursts from other stations. The receiving station detects the appropriate bursts and processes the received data [159]. TDMA is used in digital second generation (2G) cellular systems, such as Global System for Mobile Communications (GSM).

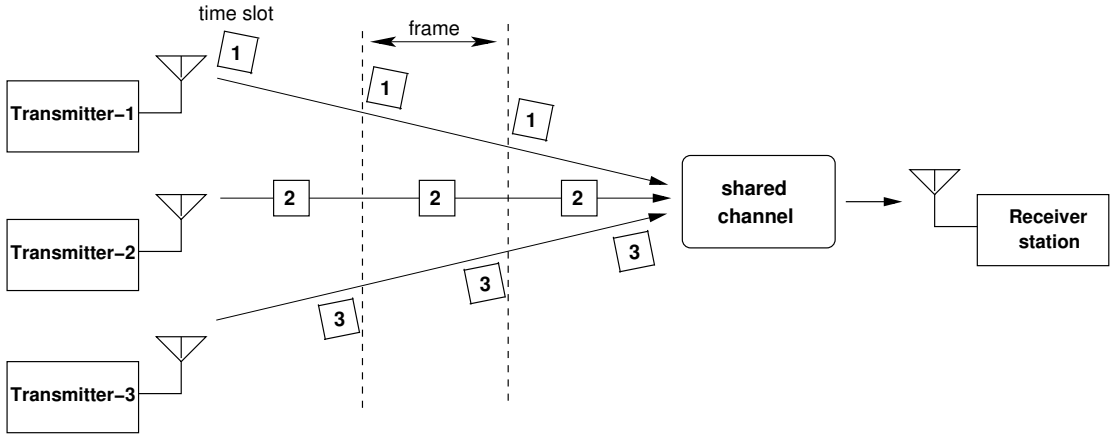


Figure 9.2.: A diagram of the TDMA approach

9.1.2. CSMA

A popular MAC protocol design, CSMA uses the carrier sense mechanism. Before transmitting a packet, the sender listens on the channel to determine if any other transmission is in progress. If it is, the sender defers, waiting until the channel becomes idle. There is a rich literature of CSMA protocols: schemes differ based on how persistently they try when the channel is idle (e.g., a node may send with probability p when the channel is idle), and in how nodes detect collisions [22, 141]. Wireless MAC protocols must handle the following problems:

1. Using suitable collision avoidance schemes to reduce the number of wasted transmissions.
2. Providing reasonable fairness among contending nodes.
3. Coping with hidden terminals.
4. Taking advantage of exposed terminals.

No existing MAC protocol successfully solves all these problems. Most practical MAC protocols favor reducing collisions over maximising every bit of available capacity [89, 98, 159].

9.2. System Model

This system has a large number of particles distributed in the environment. The particles use a shared wireless communication channel to communicate with each other and determine each others' location. They should maintain stability and must stay together. They are controlled by a leader and should converge in a certain area and distribute around the leader's position. In different scenarios, they can follow the leader to a specific position.

In terms of construction, there are many ways to connect the particles. In this work, the system is investigated by constructing a fully connected topology over shared bus. Different forms of architecture are explored [19, 20] in order to investigate the system over different performance parameters. In this topology, each particle is linked to all other particles in the flock. In other words, each particle can communicate with all other particles, creating the flock system. Particle P_0 is controlled and selected as the leader. In this part of the work, the system is constructed in two stages. The first, Communication, is modelled at a high abstraction level, meaning communication between particles is done using a shared variable, as shown in Figure (9.3). In the other stage communication is refined by the insertion of a noisy wireless communication channel, as explained later.

9.3. Communication Scenarios Based on Multichannel Access Protocols

To meet the requirements of designing an efficient communication system, multiple access communication protocols are necessary to allow multiple resources to use a shared transmission medium simultaneously, meaning these multiple access schemes are employed

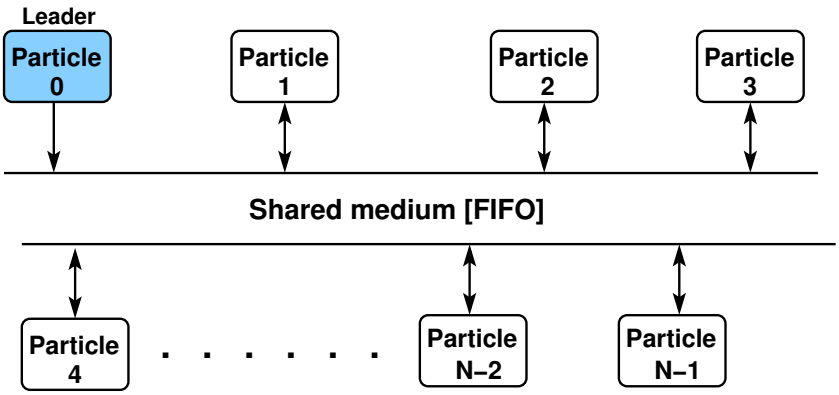


Figure 9.3.: Flocking behaviour system over shared bus

to manage multiple access issues based on a multiplexing technique. Moreover, these schemes determine how data streams can be read efficiently from each communication node, then transmitted over a shared medium and finally broadcast to all network nodes [22]. In this section we illustrate how these techniques are used to construct communication between the particles in a flocking behaviour system based on a shared medium. There are two strategies to sharing the medium: controlled access technique and contention based protocols [161]. We must show that our target is not to investigate protocol issues but rather, as mentioned above, to prove that incorporating wireless features early in the developed methodology is very advantageous. We must also show how to integrate communication modelling and design modelling early in the system development, and how this allows us to investigate the system very easily and make changes quickly.

9.3.1. TDMA Modelling

In this scenario, the system model consists of (N) particles communicating over our wireless channel model [133]. Here the wireless channel is typically shared based on TDMA. TDMA is appealing as it eliminates collision and is perfectly fair: each particle has a dedicated transmission rate of $\frac{R}{N}$ bps during each time slot, where (R) is the transmission rate of the channel [24]. As shown in Figure (9.4), the main component needed to model the TDMA technique in SystemC is sc_signal resolved [8]. It allows the particles to behave as multiple writers to access the wireless channel. The time is divided into non-overlapping time slots that are allocated to different particles. All the particles have access to the total band. In this system we assume all particles to be identical, so TDMA assigns a fixed predetermined channel time slot to each particle; it results in assigning a fraction $\frac{1}{N}$ of the total channel capacity. Program segment (9.1) illustrates how the TDMA approach is modelled in this system.

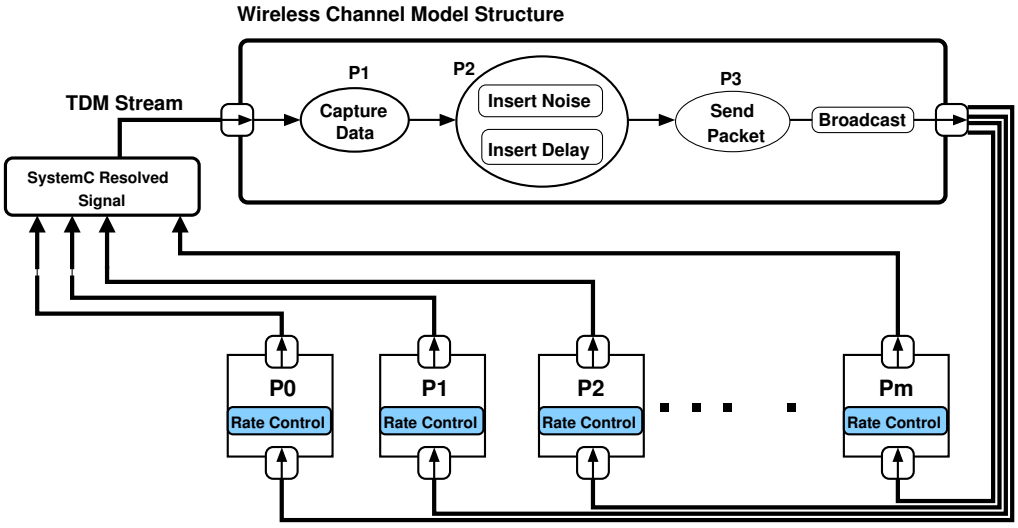


Figure 9.4.: System constructed based on TDMA scenario

Algorithm 9.1 Modelling TDMA approach

```
if(!get_node_name.compare(node_name))
{
     offset=node_index;
}
while (true)
{
....
....
if(offset%N==0)
{
     out->write(info);
     offset=N-1;
}
else
{
     offset=offset-1;
}
....
....
}
```

9.3.2. Non-persistent CSMA Modelling

The CSMA protocol is a network arbitration protocol which regulates communication between several resources that communicate over a unique channel. It is widely studied using various techniques [22]. This part of the work focuses on non-persistent CSMA

technique because of its simplicity and good performance. We assume here that the particles are not limited by transmit power, which means that each particle can transmit at any power required to reach all the other particles in the system (fully connected topology).

The system model consists of N particles distributed uniformly and communicating over the wireless channel model [133]. As shown in Figure (9.5), the non-persistent CSMA is modelled in SystemC using two main elements, the mutual exclusion object (Mutex) and sc_signal resolved [8]. The first element lets particles share the wireless channel model without colliding by allowing just one particle to send data packets; the other element allows the particles to act as multiple writers to access the wireless channel. A particle that has a packet to send senses the channel by checking the Mutex. If the channel is idle, the particle sends immediately. If the channel is busy, the particle waits a random amount of time and then senses the channel again, but, in this case, the packet is rescheduled with the new value of the current position [23]. Program segment (9.2) illustrates how CSMA approach is implemented based on SystemC in this system.

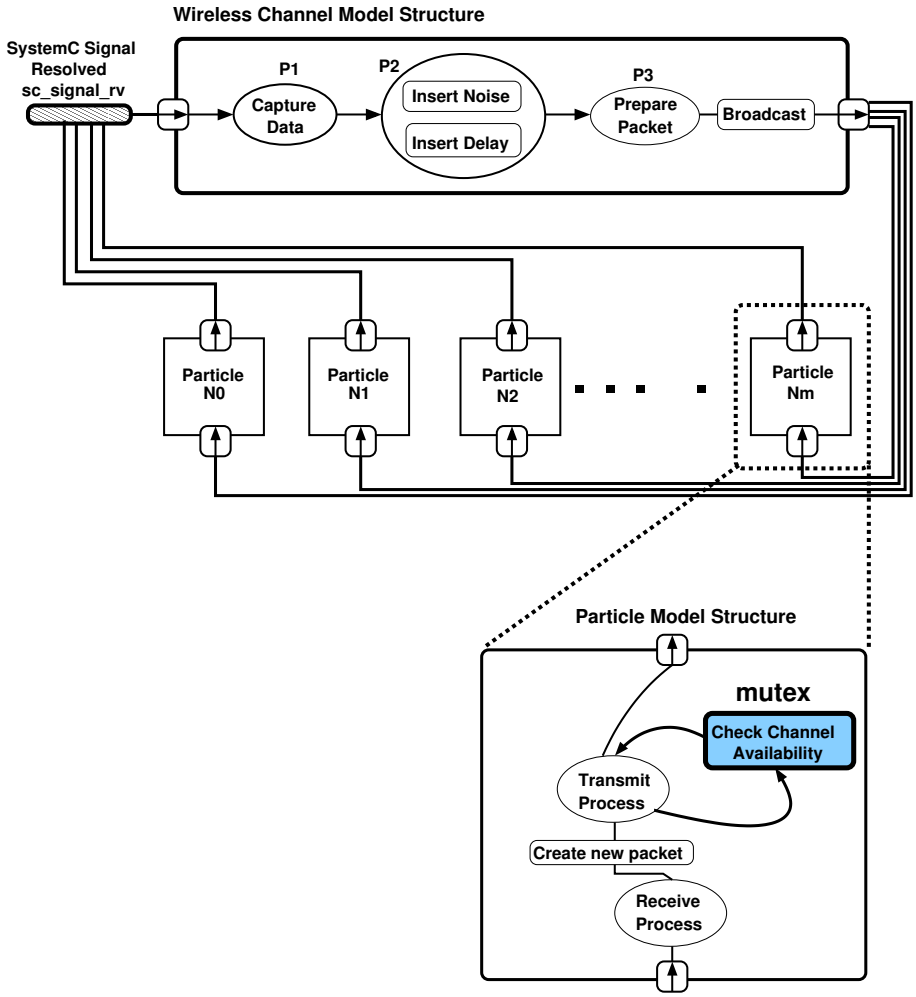


Figure 9.5.: System constructed based on CSMA scenario

Algorithm 9.2 Modelling CSMA approach

```
....
while (true)
{    randno = RNG();
     wait(randno,SC_MS);
     if(node_name.compare("Node0") )
     {    if(access->trylock()!=-1) {
          ....
          out->write(info);
          access->unlock();
          ....
     }     }
     else
     {
     out->write(info); // particle P0 is leader, so always sending
packets.
     }
....
}
```

9.4. Experimental Results

In order to simplify the simulation, only fifteen particles $(N = 15)$ are created to run the experiments. As shown in Figure (9.6), the relative positions of these particles are defined as a mesh and the leader position is located in the corner. The packet length is fixed (16 bytes). The simulation in this experiment contains the following parts: the first investigates the system behaviour and system dynamics based on the TDMA scenario; the second is based on the non-persistent CSMA scenario; and in the last part, we also investigate the effects of inserting noise and communication delay.

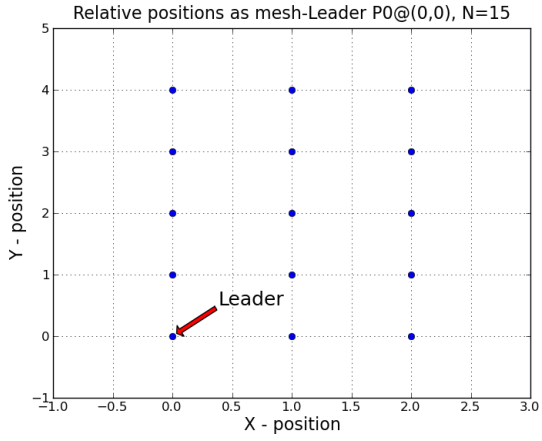
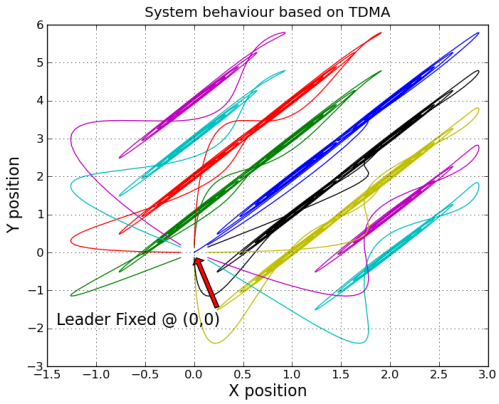


Figure 9.6.: The initial values of the relative positions

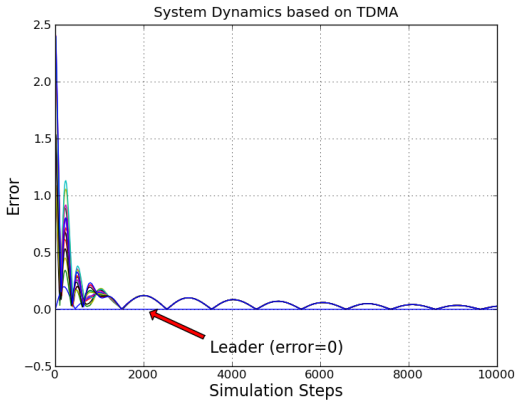
9.4.1. TDMA

i- Leader Fixed

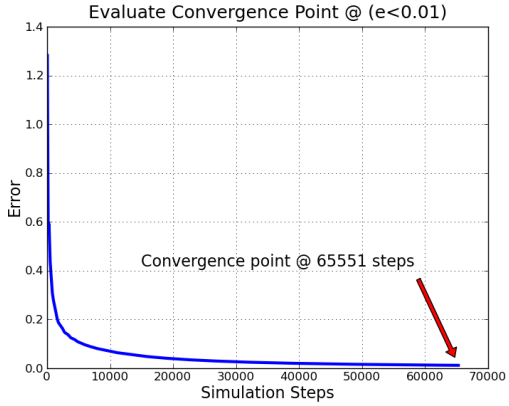
In the first stage of this experiment, the leader is fixed at position (0,0) and the other 14 particles start moving from the initial position (0,0). These particles navigate under the proposed control scheme based on the relative position values defined at the beginning of the simulation, as shown in Figure (9.6), and the system parameters indicated in Table (9.1). The particles aim to converge to the desired particles' relative positions defined at the beginning of the simulation. At the end of the simulation, the system converged and the simulation was stopped when the total error value was ($\varepsilon < 0.01$). The system behaviour (Figure (9.7-A)) clearly shows that the particles remained in the same position structure (mesh) throughout the simulation. Thus it is proven that the particles consistently and effectively avoid contact with one another. Figure (9.7-B) illustrates the RMS error and Figure (9.7-C) illustrates the convergence point of the system, which was obtained at 65,551 simulation steps.



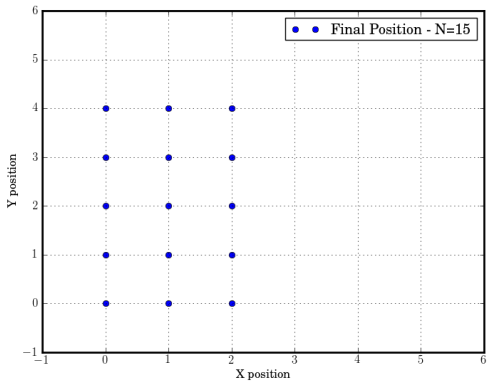
(A): System behaviour



(B): RMS error



(C): Convergence point at 0.01



(D): The final positions of the particles

Figure 9.7.: System behaviour for TDMA with fixed leader and with arrangement of Figure(9.6)

Parameters	Values
No. of Particles	15
Multiple Access Protocols	TDMA, CSMA
Transmission Speed	1 Packets/simulation step
Acceleration Range	-0.2 to 0.2
Speed Range	-1.0 to 1.0
Leader	Particle P_0
Relative Position Shape	Mesh
Error (ε)	0.01 and 0.05
Proportional Gain (Kp)	0.03
Derivative Gain (Kd)	0.1
Desired Distance (L_d)	1

Table 9.1.: System Parameters

ii- Leader Moved

In the second case, the leader is assumed to be moveable and the other particles will follow the leader to the specific target, as shown in Figure (6.16). The relative position values are defined as mesh (Figure (9.6)). In the beginning of the simulation, the leader and all the other particles start moving from initial position (0,0), as shown in Figure (9.8-A). The values of the parameters in this simulation are the same as previously indicated in Table (9.1). In the first phase of this experiment, the leader is assumed to be moved into position (20,20), then changes direction into position (50,-10) and finally into the target position at (180,120). This leader's path is selected as the same path that the leader follows in Chapters Six and Seven (Figure (6.16)), because we can easily compare between the different communication approaches used in this system.

Screen-shots A-F of Figure (9.8) show the particles' evolution in position and the achievement of flocking motion. In screen-shots A-E, position (0,0) denotes the starting point of all particles and also shows how the particles follow the leader. In the last screen-shot, F, the leader approaches the target position and all the particles follow it and converge around the target. Figures (9.9-A, B) illustrate the positional error curve and RMS error respectively, while Figure (9.9-C) illustrates the convergence point of the system, which was obtained when $\varepsilon < 0.05$. The final positions of the particles are illustrated in Figure (9.9-D).

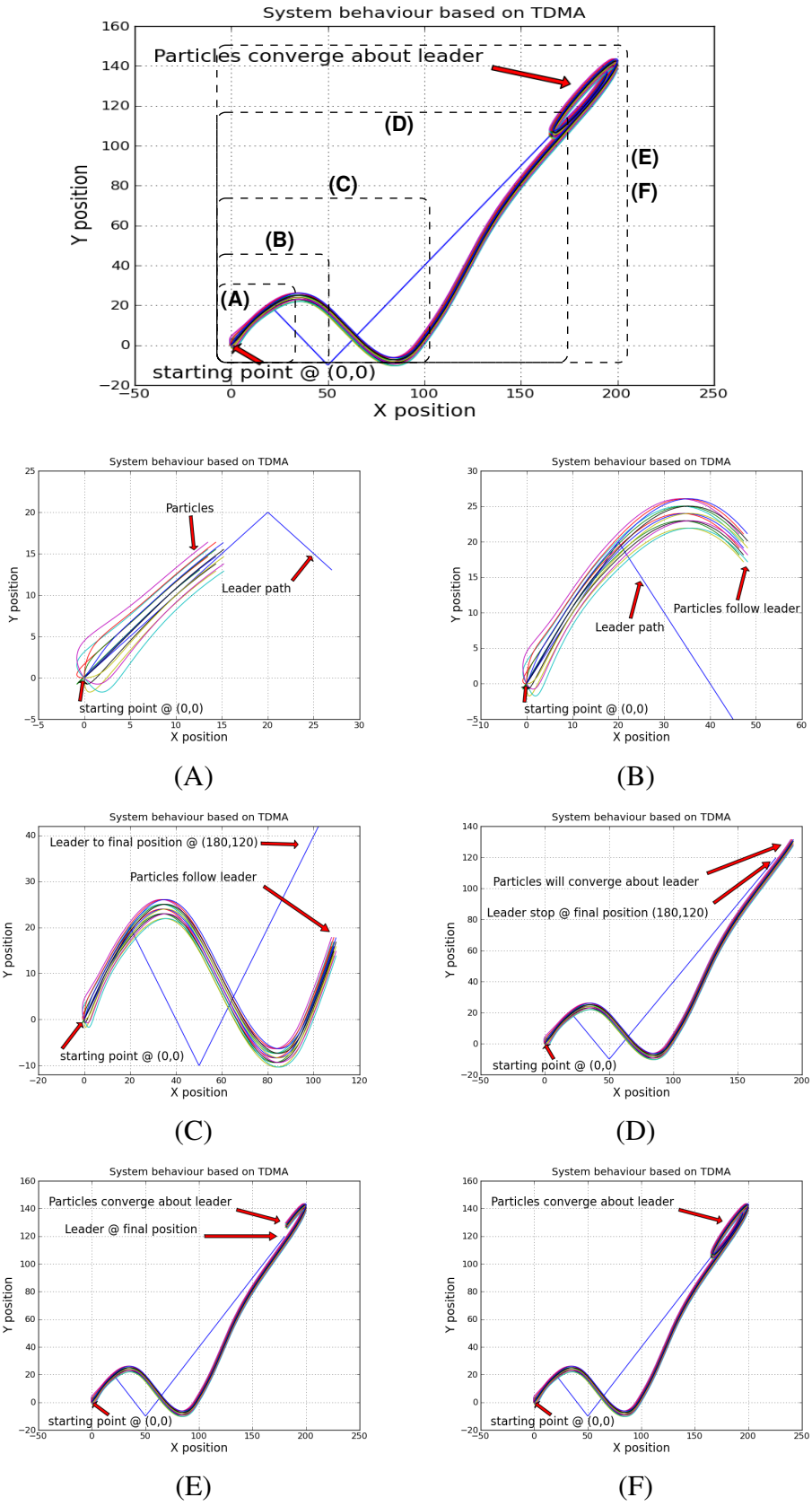
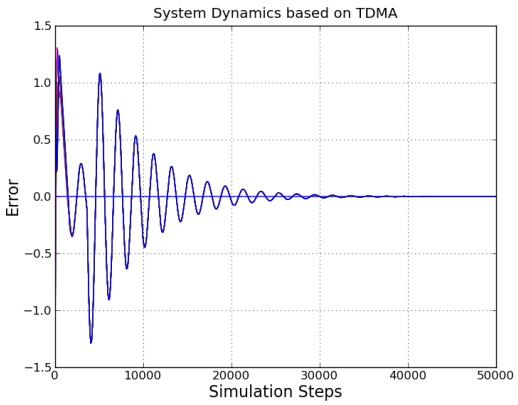
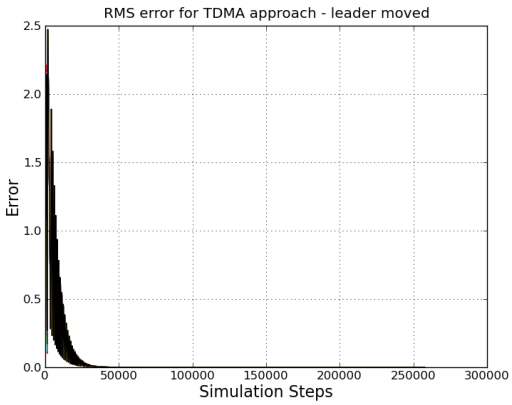


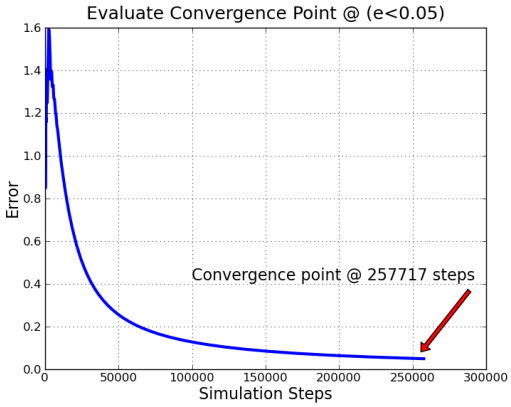
Figure 9.8.: Details of system behaviour for TDMA approach with moved leader and with arrangement of Figure (9.6)



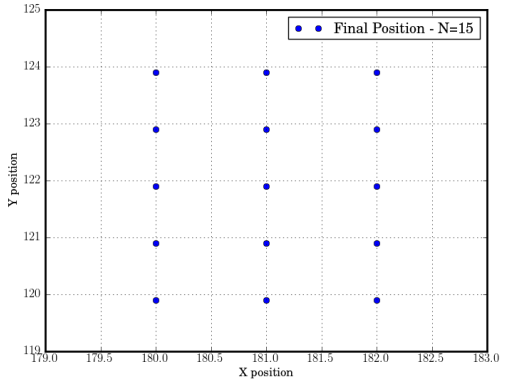
(A): Positional error



(B): RMS error



(C): Convergence point at 0.05



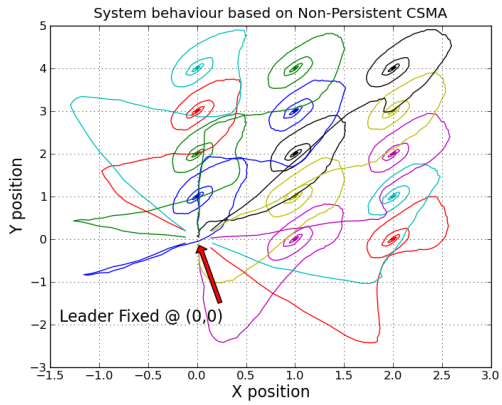
(D): The final positions of the particles

Figure 9.9.: System behaviour for TDMA with moved leader

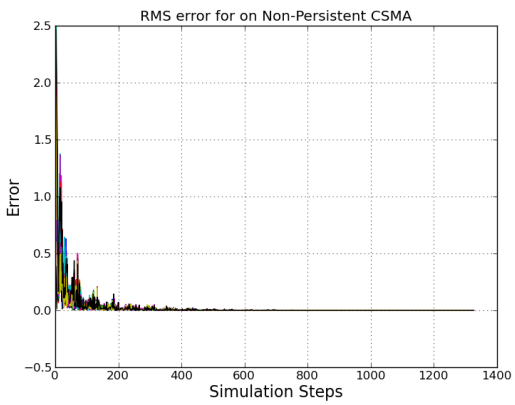
9.4.2. Non-persistent CSMA

i- Leader Fixed

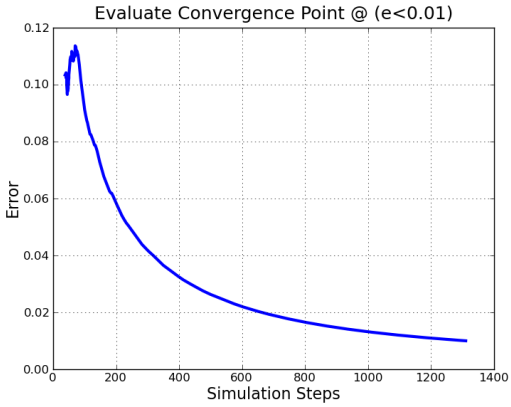
In the second part of the simulation, the particles communicate using a non-persistent CSMA scheme. The experiment is done under the same conditions illustrated above, with the system parameters indicated in Table (9.1 on page 145). The system behaviour shown in Figure (9.10-A) is most likely obtained from the TDMA scenario, except that simulation time is less than in the TDMA experiment. The RMS error in Figure (9.10-B) illustrates the simulation time, and when the system convergence is shown in Figure (9.10-C). The final position of the particles is illustrated in Figure (9.10-D).



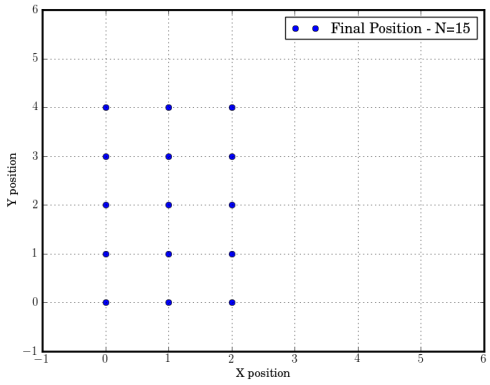
(A): System behaviour



(B): RMS error



(C): Convergence point at 0.01



(D): The final positions of the particles

Figure 9.10.: System behaviour for non-persistent CSMA approach with fixed leader

ii- Leader Moved

In the second case, the leader is assumed to be moveable and the other particles will follow the leader to the specific target, as shown in Figure (9.11 on the next page), and the system uses the same initial parameter values as shown by Table (9.1). All particles' initial positions are at original point $(0,0)$ and velocities are based on the error value of each particle. The environment is assumed to have an identical effect on all particles. Again the leader's path is selected as the same path that the leader follows in Chapters Six and Seven (Figure (6.16)), in order to compare the results of all the cases.

Figure (9.11 on the following page) depicts screen-shots A-F show the particles' trajectories on the $x-y$ plane and the achievement of flocking motion. The blue straight line in the main figure represents the leader's path, while the other lines represent the particles' paths. Also, Figure (9.12 on page 150-C) illustrates the convergence of the particles' positions; it is clear that all particles converge to the positions defined at the beginning of the simulation (Figure (9.12 on page 150-D)).

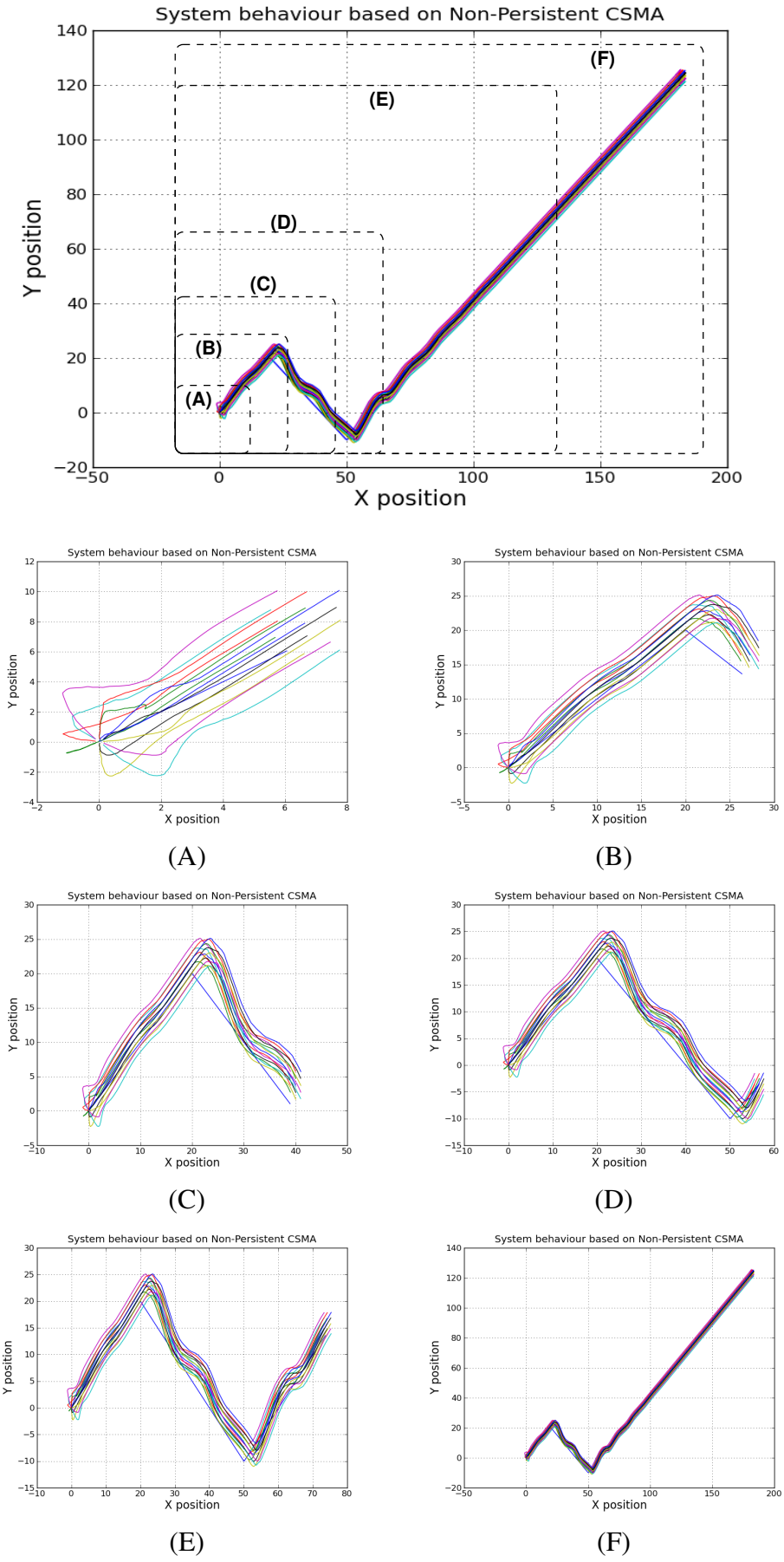
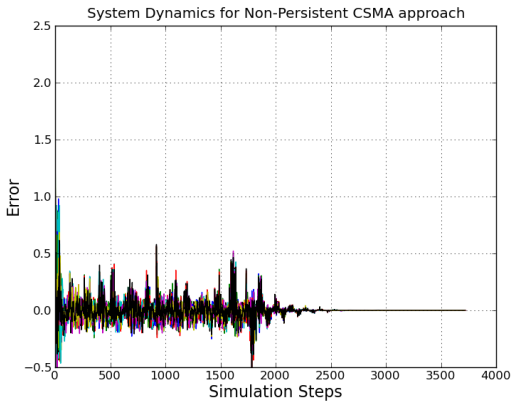
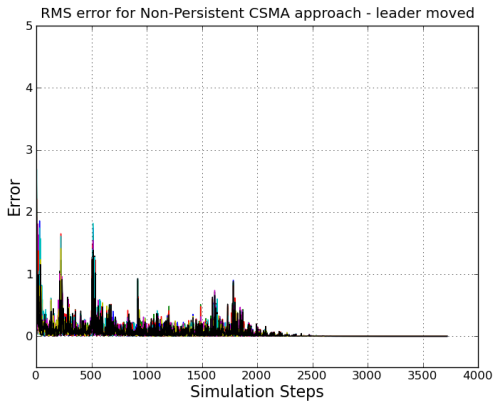


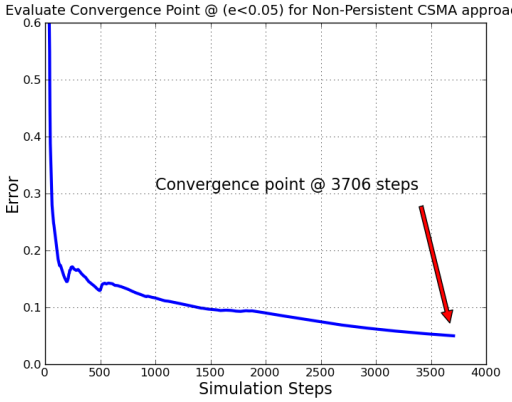
Figure 9.11.: Details of system behaviour for CSMA approach with moved leader and with arrangement of Figure (9.6)



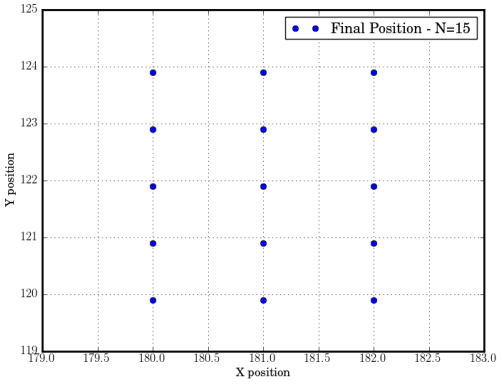
(A): Positional error



(B): RMS error



(C): Convergence point at 0.05



(D): The final positions of the particles

Figure 9.12.: System dynamics for CSMA, movable leader

In this work, the most important finding of this chapter constitute more proof that wireless features are incorporated successfully into SystemC design methodology, which can be used from now to model and design complex wireless communication systems. TDMA and non-persistent CSMA protocols have been modelled before, but the ways and purposes of modelling these protocols are unlike ours, because our aim here is not to investigate these protocols but to prove that inserting wireless communication into the developed methodology is very advantageous. The final results have been validated and it has been proven that communication has a big impact on system dynamics, i.e., small changes in the wireless specifications create big changes in the system dynamics.

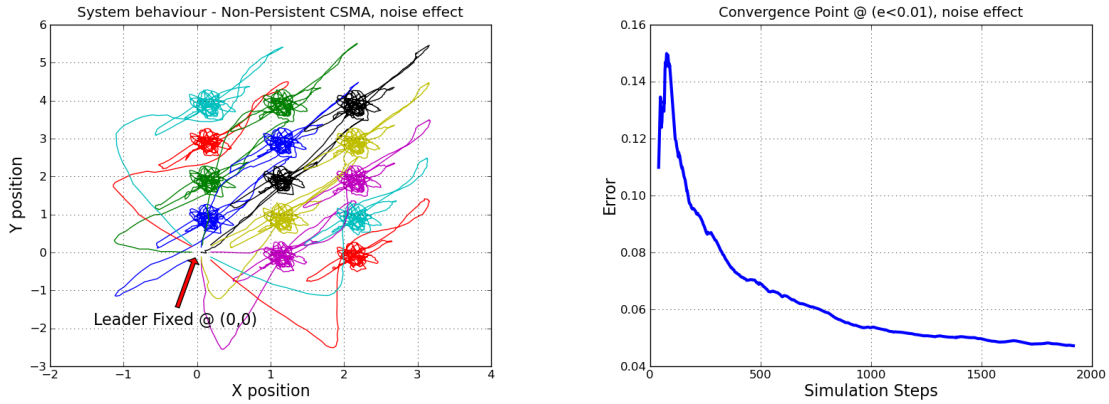
9.4.3. Impact of Noise

In this section we investigate the effect of inserting noise on system behaviour and system performance. We consider a case of the system modelled in section 9.4.2, which is

Algorithm 9.3 Inserting impulsive Noise

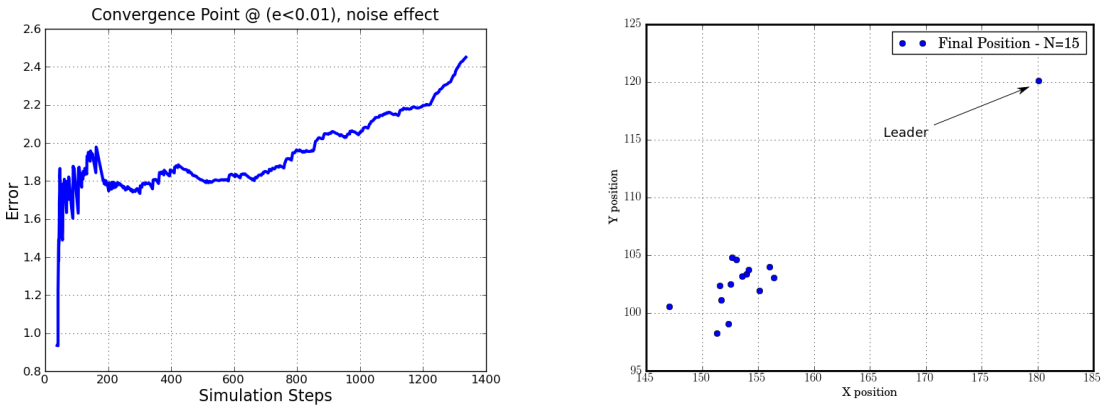
```
...
lamda = 0.01;
next_error_position = rng.exponential(lamda);
while(true)
{
...
packet_counter++;
if (packet_counter >= next_error_position) {
     packet_counter =0;
     i_frame = ~i_frame;
     next_error_position = rng.exponential(lamda);
...
}
...
}//end while
```

modelled based on the CSMA approach. We consider independent impulsive noise as random forces depending on exponential distribution to act on individual particles. The train of noise impulses can be regarded as a renewal process, i.e. a series of random events in which the intervals between events are independent and identically distributed. Inter-arrival times between impulses exhibit statistical behaviour depending on exponential distribution as illustrated in the program segment (9.3). The effects of the impulsive noise on system behaviour are shown in Figure (9.13-A), while Figure (9.13-B) illustrates the system converging, which indicates the system is taken a more time to get converge when compare to the case without inserting the noise. Hence this experiment gives us sufficient conditions for the considered system to achieve flocking behaviour system based on the CSMA approach in a noisy environment.



(A) System behaviour (B) System convergence

Figure 9.13.: Effect of noise on the system based on non-persistent CSMA approach with fixed leader



(A) (B)

Figure 9.14.: System convergence and final position of the particles

In the second case, the leader is movable. Figure (9.15 on the next page) illustrates a flock with $(N = 15)$ moving in a $2D$ linear environment under the effect of noise. The figure shows that the distance between the particles and the leader is greater than that of Figure (9.11 on page 149), because some packets sent from the leader are distorted, which leads the particles to change their path, as indicated in Figure (9.15 on the next page-C, G). But once the particles receive the correct packet, they try to converge again to the leader. Screen-shots A-H of Figure (9.15 on the following page) show the positions of the particles at several simulation steps. It can also be seen in Figure (9.14-A, B) that the particles do not converge to the leader's path (desired path) and then to the final positions.

9.5. Summary

In this chapter, the developed methodology is applied to the model of a flocking behaviour system selected as a case study in order to validate it. The communication between the particles is addressed based on a shared channel with two scenarios. The first is TDMA, which is a controlled access technique, and the other is non-persistent CSMA, which is a contention-based protocol. Modelling TDMA and CSMA protocols with SystemC allows us to build up software components that are ready-to-use in wireless development prototyping. These software components can be conveniently used as a substitution for hardware and can consequently reduce costs.

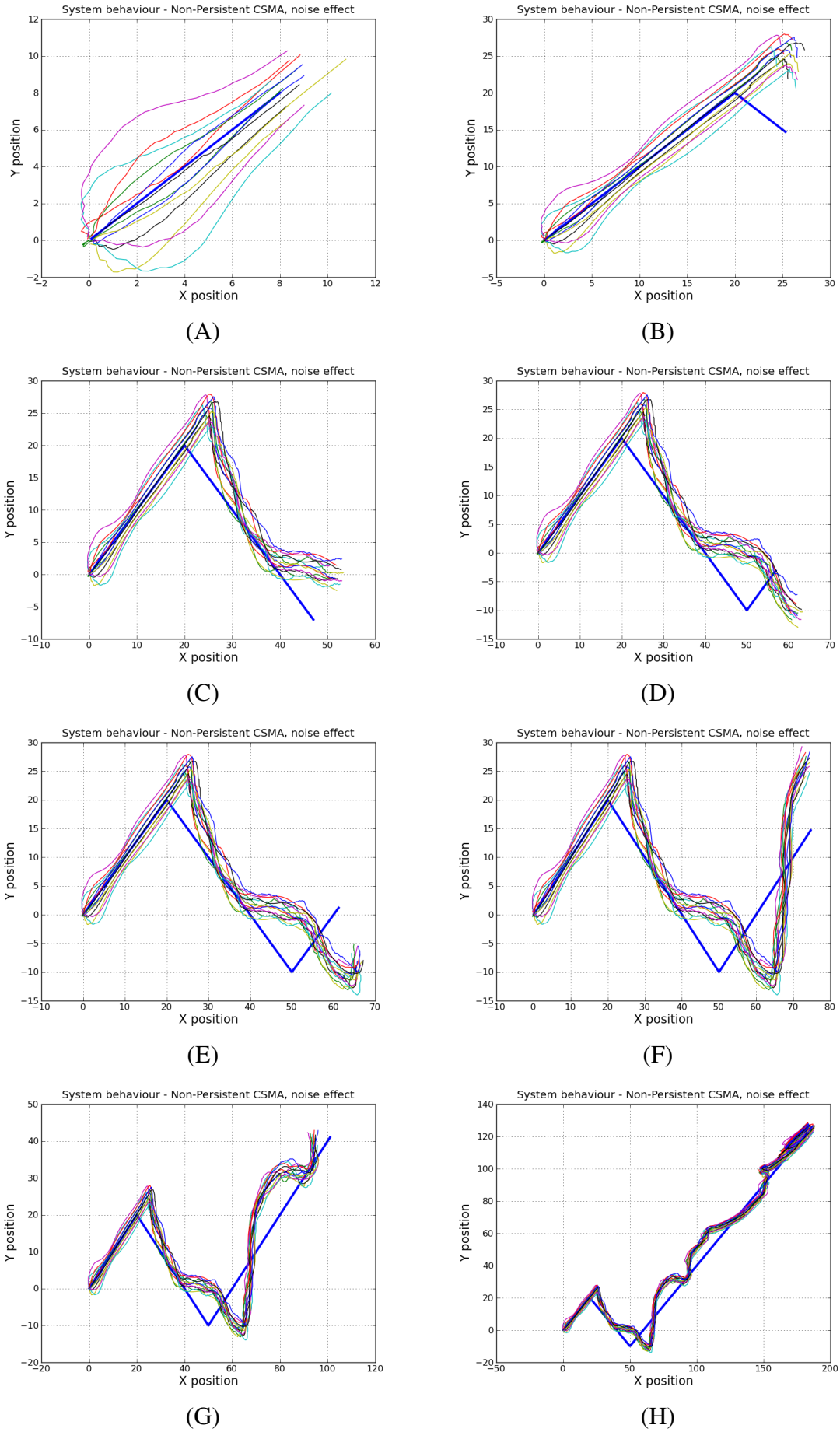


Figure 9.15.: Details of system behaviour of Figure (9.11) under noise effect

10. Conclusions

This research illustrates how wireless features can be incorporated into the existing SystemC design methodology, including an efficient simulation of wireless systems. To date, the SystemC modelling language lacks a standard framework that supports modelling of wireless communication systems (in particular the use of wireless communication channels). The components to be investigated in order to achieve this target are divided into three parts: developing a system-level model of a digital wireless communication channel that represents the core of any communication system; creating a small library of dedicated elements at system level, such as PLL (Phased Locked Loop), 8B/10B Encoder/Decoder and several modelling communication protocols; and concluding with a case study of a flocking behaviour system to validate the wireless extension methodology.

Hence, it is very important for designers to develop a SystemC environment to support wireless, because SystemC provides a consistent methodology and a homogeneous design flow for the design and refinement of complex digital systems; it is also essential to manage complexity and enhance designer productivity. Therefore, once we have incorporated wireless features into existing SystemC environments, we can investigate the changing behaviour of complex wireless communication systems very quickly, allowing us to implement the system as we wish, view designs at different levels of abstraction and evaluate system performance early in the design cycle (early modelling). All this can be done because SystemC is a unified environment, which means everything can be modelled in the same platform.

10.1. Summary of contributions

In this research, we present how existing SystemC design methodology can be developed to support wireless communications systems. The work is done based on a number of stages that are summarised as follows:

- As the first phase of this research, the modelling of a noisy digital communication channel in SystemC is presented. This model is the most important model needed

to extend SystemC methodology to support wireless systems, because it represents what is missing in the SystemC language to support wireless systems. This work demonstrates a simple and computationally efficient way to model a communication channel within a system level. The model is developed at a high level of abstraction, which allows for rapid simulation and early estimation, which are necessary for successful system development using the SoC design methodology. For instance, the wireless communication channel model becomes a part of the developed methodology and can then be inserted into any system.

- We have also presented an RTL-level SystemC model of an 8B/10B Encoder/Decoder core. The use of 8B/10B coding is an important technique in the construction of high performance serial interfaces. Moreover, as systems become more tightly integrated (as in, for example, SoC) the ability to evaluate system performance at early stages of design becomes increasingly important. This is facilitated by the SystemC design methodology, and by following an IP-based design.

- We present a development of existing SystemC design methodology that can be used to model wireless systems. The developed methodology is proposed based on two approaches: a hierarchical partitioning of steps in the System/Communication aspects, or a leading into Naïve Structure, which means one partitioning step. In both cases, Communication comes to be a new branch of the System aspect in traditional SystemC methodology.

- In the third stage of this research, the developed methodology is applied to the model of a flocking behaviour system selected as a case study, in order to validate it. The communication aspects are modelled based on a P2P link, which means the wireless channel between a pair of particles starts to look like a link. The final results of the modelled system have been validated and it has been proven that communication has a big impact in system dynamics, i.e., small changes in the wireless specifications create big changes in the system dynamics.

- After that, the work is taken in another direction of research, investigating the interaction between communication and system stability; we evaluate how the system parameters affect system behaviour. we investigate how the stability of the flocking behaviour system is optimised using some important concepts of graph theory. The system has been modelled successfully; positive results representing system behaviour and system dynamics were established.

- On the other hand, in order to investigate the system under different communication approaches that support MAC protocols, the communication between particles is addressed based on a shared channel with two scenarios: TDMA and non-persistent

CSMA. Modelling TDMA and CSMA protocols with SystemC allowed us to build up software components that are ready-to-use in wireless development prototyping. Our aim here is not to investigate these protocols but to prove that inserting wireless communication into the developed methodology is very advantageous. The final results of the modelled system have been validated and it has been proven that communication has a big impact on system dynamics.

Therefore, incorporating wireless features into existing SystemC design methodology (as done in this research) is a very important task, because by developing SystemC as a design tool to support wireless systems, HW, SW and communication can be modelled, refined and validated simultaneously on the same platform, and the design space expanded into a two-dimensional design space: system and communication.

10.2. Future Work

There are different ways to extend the research work that has been presented in this thesis as following:

- For example, an FPGA could be used for 8B/10B encoder/decoder model that covered in Chapter Three. RTL-level model of an 8B/10B encoder/decoder block could be converted to a hardware description language such as VHDL or Verilog to build an application and test it on FPGA.

- Another possibility for future research would be considered is required to complete the model of the wireless channel using SystemC by modelling more wireless features such as fading effects and other noise types in order to create more efficient simulations.

- Finally, we can try to explore efficient SystemC to RTL translators. Currently available tools like the SC2V translator do not support translation of all SystemC constructs. Adding capabilities of modeling Thread and Clocked Thread processes to existing tools would greatly enhance design productivity and time to market.

Appendix A : 8B/10B Encoder - Decoder RTL Model

This appendix contains the SystemC source code for the RTL model of 8B/10B Encoder Decoder. The model was developed and compiled using the SystemC 2.0.1 library. All model simulations were done using the reference simulator provided by OSCI.

8B/10B Encoder Model

File: encoder.h

```
#include "systemc.h"

SC_MODULE (enc_eight_ten) {

sc_in<sc_logic> RESET;

sc_in<bool> SBYTECLK;

sc_in<sc_logic> KI;

sc_in<sc_logic> AI,BI,CI,DI,EI,FI,GI,HI;

sc_out<sc_logic> AO,BO,CO,DO,EO,IO,FO,GO,HO,JO;

// Signals to tie things together

sc_signal<sc_logic> XLRESET, LRESET;

sc_signal<sc_logic> L40 ,L04, L13, L31, L22;

sc_signal<sc_logic> F4, G4, H4, K4, S, FNEG;

sc_signal<sc_logic> PD1S6, ND1S6, PD0S6, ND0S6;
```

```
sc_signal<sc_logic> ND1S4, ND0S4, PD1S4, PD0S4;

sc_signal<sc_logic> COMPLS4, COMPLS6, NDL6;

sc_signal<sc_logic> PDL6, LPDL6, PDL4, LPDL4;

sc_signal<sc_logic> NA0, NB0, NC0, ND0, NE0, NI0;

sc_signal<sc_logic> NF0, NG0, NH0, NJ0, SINT;

//PROCESS: SYNCRST; Synchronize and delay RESET one clock for startup

void SYNCRST();

void enc_5b(void);

void FN3B(void);

void FNS (void);

void disparity(void);

void CMPLS4(void);

void CMPLS6(void);

// PROCESS: ENC5B6B; Generate and latch LS 6 encoded bits

void ENC5B6B(void);

void ENC3B4B (void);

// Constructors

SC_CTOR(enc_eight_ten) {

SC_THREAD(SYNCRST);

sensitive << XLRESET << SBYTECLK << RESET;

dont_initialize();

SC_THREAD(enc_5b);

sensitive << KI << SBYTECLK;

dont_initialize();

SC_THREAD(FN3B);
```

```
sensitive << SBYTECLK << FI << GI << HI << KI;

dont_initialize();

SC_THREAD(FNS);

sensitive << LRESET << SBYTECLK << PDL6 << L31 << DI << EI << NDL6 << L13;

dont_initialize();

SC_THREAD(disparity)

sensitive << FNEG << F4 << G4 << H4 << SBYTECLK;

dont_initialize();

SC_THREAD(CMPLS4);

sensitive << LRESET << SBYTECLK << PDL6;

dont_initialize();

SC_THREAD(CMPLS6);

sensitive << LRESET<< SBYTECLK << PDL4;

dont_initialize();

SC_THREAD(ENC5B6B);

sensitive << LRESET << SBYTECLK << COMPLS6;

dont_initialize();

SC_THREAD(ENC3B4B);

sensitive << LRESET << SBYTECLK << COMPLS4;

dont_initialize();

} // End of Constructor

}; // End of Module
```

File: encoder.cpp

```
#include "enc_8b10b.h"

void enc_eight_ten::SYNCRST(){

while(true) {

wait();

if(SBYTECLK.posedge()){

XLRESET = RESET ;

}else if (SBYTECLK.negedge()){

LRESET = XLRESET ;

}

}//while

} //end process SYNCRST

//

// 5b Input Function (Reference:  Figure 3)

//

void enc_eight_ten::enc_5b(void) {

while(true) {

wait(1); // Wait for the event in sensitivity list to occure

// Four 1's

L40.write(AI & BI & CI & DI); // 1,1,1,1

wait(SC_ZERO_TIME);

// Four 0's

L04.write( ~((sc_logic)AI) & ~((sc_logic)BI) & ~((sc_logic)CI) & ~((sc_logic)DI));

wait(SC_ZERO_TIME);

// One 1 and three 0's
```

```
L13.write((~((sc_logic)AI) & ~((sc_logic)BI) & ~((sc_logic)CI) & DI)
|(~((sc_logic)AI) & ~((sc_logic)BI) & CI & ~((sc_logic)DI))
|(~((sc_logic)AI) & BI & ~((sc_logic)CI) & ~((sc_logic)DI))
|( AI & ~((sc_logic)BI) & ~((sc_logic)CI) & ~((sc_logic)DI)));
wait(SC_ZERO_TIME);
// Three 1's and one 0
L31.write((AI & BI & CI & ~((sc_logic)DI))
|(AI & BI & ~((sc_logic)CI) & DI)
|(AI & ~((sc_logic)BI) & CI & DI)
|(~((sc_logic)AI) & BI & CI & DI));
wait(SC_ZERO_TIME);
// Two 1's and two 0's
L22.write((~((sc_logic)AI) & ~((sc_logic)BI) & CI & DI)
|(~((sc_logic)AI) & BI & CI & ~((sc_logic)DI))
|(AI & BI & ~((sc_logic)CI) & ~((sc_logic)DI))
|(AI & ~((sc_logic)BI) & ~((sc_logic)CI) & DI)
|(~((sc_logic)AI) & BI & ~((sc_logic)CI) & DI)
|(AI & ~((sc_logic)BI) & CI & ~((sc_logic)DI)));
wait(2);
}//while
} // End of function send_data
// PROCESS: FN3B; Latch 3b and K inputs
void enc_eight_ten::FN3B(void){
while(true) {
wait(1);
```

```
// Falling edge of clock latches F,G,H,K inputs

if(SBYTECLK.negedge()){

F4.write( FI );

G4.write( GI );

H4.write( HI );

K4.write( KI );

wait(SC_ZERO_TIME);

FNEG.write( F4 ^ G4 );

wait(2);

}

}//while

}// end process FN3B

// PROCESS: FNS; Create and latch "S" function

void enc_eight_ten::FNS (void){

while(true) {

wait(); // Wait for the event in sensitivity list to occure

if (LRESET == SC_LOGIC_1){

S.write( SC_LOGIC_0 );

}else if (SBYTECLK.posedge()) {

wait(SC_ZERO_TIME);

S.write( (PDL6 & L31 & DI & ~((sc_logic)EI))

|(NDL6 & L13 & EI & ~((sc_logic)DI)));

}// end if

wait(2);

}//while
```

```
}// end process FNS

//Intermediate term for "F4 is Not Equal to G4"

void enc_eight_ten::disparity(void)

{

while(true) {

wait(1);

PD1S6.write( (~((sc_logic)L22) & ~((sc_logic)L31) & ~((sc_logic)EI))|(L13 & DI
& EI));

ND1S6.write( (L31 & ~((sc_logic)DI) & ~((sc_logic)EI))|(EI & ~((sc_logic)L22)
& ~((sc_logic)L13))| K4);

PD0S6.write( (~((sc_logic)L22) & ~((sc_logic)L13) & EI)| KI);

ND0S6.write( (~((sc_logic)L22) & ~((sc_logic)L31) & ~((sc_logic)EI)) |(L13 &
DI & EI));

ND1S4.write( (F4 & G4));

ND0S4.write( (~((sc_logic)F4) & ~((sc_logic)G4)));

PD1S4.write( (~((sc_logic)F4) & ~((sc_logic)G4)) | (FNEG & K4));

PD0S4.write( (F4 & G4 & H4));

wait(SC_ZERO_TIME);

PDL6.write((PD0S6 & ~((sc_logic)COMPLS6))

| (COMPLS6 & ND0S6)

| (~((sc_logic)ND0S6) & ~((sc_logic)PD0S6) & LPDL4));

wait(SC_ZERO_TIME);

NDL6.write(~((sc_logic)PDL6));

wait(SC_ZERO_TIME);

PDL4.write((LPDL6 & ~((sc_logic)PD0S4) & ~((sc_logic)ND0S4))

|(ND0S4 & COMPLS4)
```

```
| (~((sc_logic)COMPLS4) & PDOS4)) ;

}//while

}

void enc_eight_ten::CMPLS4(void){

while(true) {

wait(1); // Wait for the event in sensitivity list to occure

if (LRESET == SC_LOGIC_1){

LPDL6.write(SC_LOGIC_0);

wait(SC_ZERO_TIME);

}else if (SBYTECLK.posedge()) {// Rising edge

LPDL6.write(PDL6); // ..  latches S4

wait(SC_ZERO_TIME);

}

COMPLS4.write((PD1S4 & ~( (sc_logic)LPDL6)) ^ (ND1S4 & LPDL6));

wait(SC_ZERO_TIME);

}//while

} //end process CMPLS4 ;

void enc_eight_ten::CMPLS6(void){

while(true) {

wait(2);

if (LRESET == SC_LOGIC_0){

LPDL4.write( SC_LOGIC_0 );

}else if (SBYTECLK.negedge()){ // Falling edge

LPDL4.write(PDL4); //..  latches S6

}
```

```
wait(SC_ZERO_TIME);

COMPLS6.write( (ND1S6 & LPDL4)^ (PD1S6 & ~((sc_logic)LPDL4)));

wait(SC_ZERO_TIME);

}//while

}// end process CMPLS6;

// PROCESS: ENC5B6B; Generate and latch LS 6 encoded bits

void enc_eight_ten::ENC5B6B(void){

while(true) {

wait(2);

NAO.write(AI);

NBO.write( LO4 |(BI & ~((sc_logic)L40)));

NCO.write( CI | LO4 | (L13 & DI & EI));

NDO.write( DI & ~((sc_logic)L40));

NEO.write( (EI & ~((sc_logic)(L13 & DI & EI))) | (L13 & ~((sc_logic)EI)));

NIO.write( (L22 & ~((sc_logic)EI)) | (LO4 & EI) | (L13 & ~((sc_logic)DI) & EI)
| (L40 & EI) | (L22 & KI)) ;

if (LRESET == SC_LOGIC_1){

AO.write(SC_LOGIC_0);

BO.write(SC_LOGIC_0);

CO.write(SC_LOGIC_0);

DO.write(SC_LOGIC_0);

EO.write(SC_LOGIC_0);

IO.write(SC_LOGIC_0);

}else if (SBYTECLK.posedge()){

AO.write( COMPLS6 ^ NAO );
```

```
BO.write( COMPLS6 ^ NBO );

CO.write( COMPLS6 ^ NCO );

DO.write( COMPLS6 ^ NDO );

EO.write( COMPLS6 ^ NEO );

IO.write( COMPLS6 ^ NIO );

}

wait(SC_ZERO_TIME);

}//while

} //end process ENC5B6B;

// PROCESS: ENC3B4B; Generate and latch MS 4 encoded bits

void enc_eight_ten::ENC3B4B (void){

while(true) {

wait(2); // Wait for the event in sensitivity list to occure

SINT.write( (S & F4 & G4 & H4) | (K4 & F4 & G4 & H4));

NFO.write( (F4 & ~((sc_logic)SINT)));

NGO.write( G4 | (~((sc_logic)F4) & ~((sc_logic)G4) & ~((sc_logic)H4)));

NHO.write( H4 );

NJO.write( SINT | (FNEG & ~((sc_logic)H4)));

if (LRESET == SC_LOGIC_1){

FO.write(SC_LOGIC_0);

GO.write(SC_LOGIC_0);

HO.write(SC_LOGIC_0);

JO.write(SC_LOGIC_0);

}else if (SBYTECLK.posedge()){

FO.write( COMPLS4 ^ NFO );
```

```
GO.write( COMPLS4 ^ NGO );

HO.write( COMPLS4 ^ NHO );

JO.write( COMPLS4 ^ NJO );

}

wait(SC_ZERO_TIME);

}//while

}//end process ENC3B4B ;
```

8B/10B Decoder Model

File: decoder.h

```
#include "systemc.h"

SC_MODULE(dec_8b10b){

sc_in<sc_logic> RESET;

sc_in<bool> RBYTECLK;

sc_in<sc_logic> AI, BI, CI, DI, EI, II;

sc_in<sc_logic> FI, GI, HI, JI;

sc_out<sc_logic> KO;

sc_out<sc_logic> HO, GO, FO, EO, DO, CO, BO,AO;

// Signals to tie things together

sc_signal<sc_logic> ANEB, CNED, EEI, P13, P22, P31;

sc_signal<sc_logic> IKA, IKB, IKC;

sc_signal<sc_logic> XA, XB, XC, XD, XE;

sc_signal<sc_logic> OR121, OR122, OR123, OR124, OR125, OR126, OR127;

sc_signal<sc_logic> XF, XG, XH;
```

```
sc_signal<sc_logic> OR131, OR132, OR133, OR134, IOR134;

// events declaration

sc_event output;

void DEC_6B_IN(void);

void DEC5B(void);

void DEC3B ( void );

// Module Constructor start here

SC_CTOR(dec_8b10b) {

SC_THREAD(DEC_6B_IN);

sensitive << RBYTECLK;

dont_initialize();

SC_THREAD(DEC5B);

sensitive << RESET << RBYTECLK << XA << XB << XC << XD << XE << AI << BI << CI
<< DI << EI;

dont_initialize();

SC_THREAD(DEC3B);

sensitive << RESET<< RBYTECLK<< XF << XG << XH << FI<< GI<< HI;

dont_initialize();

} // End of Constructor

}; // End of Module TransmitReceive
```

File: decoder.cpp

```
#include "8b10b_dec.h"

void dec_8b10b::DEC_6B_IN(void) {

while(true) {
```

```
wait(1);

// One 1 and three 0's

P13.write((ANEB & (~((sc_logic)CI) & ~((sc_logic)DI)))

| (CNED & (~((sc_logic)AI) & ~((sc_logic)BI))) );

P31.write((ANEB & CI & DI)

| (CNED & AI & BI) );

// Two 1's and two 0's

P22.write((AI & BI & (~((sc_logic)CI) & ~((sc_logic)DI)))

| (CI & DI & (~(((sc_logic)AI) & ~(sc_logic)BI)))

| (ANEB & CNED)) ;

// Intermediate term for "AI is Not Equal to BI"

ANEB.write( AI ^ BI );

// Intermediate term for "CI is Not Equal to DI"

CNED.write( CI ^ DI) ;

// Intermediate term for "E is Equal to I"

EEI.write( ~((sc_logic)(EI ^ II))) ;

IKA.write( (CI & DI & EI & II)

| (~((sc_logic)CI) & ~((sc_logic)DI) & ~((sc_logic)EI) & ~((sc_logic) II)) );

IKB.write( P13 & (~((sc_logic)EI) & II & GI & HI & JI)) ;

IKC.write( P31 & (EI & ~((sc_logic)II) & ~((sc_logic)GI) & ~((sc_logic)HI) &
~((sc_logic)JI)) );

if( RESET == SC_LOGIC_1){

KO.write( SC_LOGIC_0 );

}else if( RBYTECLK.negedge() ){

KO.write( IKA | IKB | IKC);
```

```
} //end if

// Logic for complimenting F,G,H outputs

OR131.write( (GI & HI & JI)

| (FI & HI & JI)

| (IOR134));

OR132.write( (FI & GI & JI)

| (~((sc_logic)FI) & ~((sc_logic)GI) & ~((sc_logic)HI))

| (~((sc_logic)FI) & ~((sc_logic)GI) & HI & JI));

OR133.write( (~((sc_logic)FI) & ~((sc_logic)HI) & ~((sc_logic)JI))

| (IOR134)

| (~((sc_logic)GI) & ~((sc_logic)HI) & ~((sc_logic)JI)) );

OR134.write( (~((sc_logic)GI) & ~((sc_logic)HI) & ~((sc_logic)JI))

| (FI & HI & JI)

| (IOR134) );

IOR134.write( (~((sc_logic)(HI & JI)))

& (~((sc_logic) (~((sc_logic) HI) & ~((sc_logic) JI))))

& (~((sc_logic)CI) & ~((sc_logic)DI) & ~((sc_logic)EI) & ~((sc_logic)II)) );

XF.write( OR131

| OR132 );

XG.write( OR132

| OR133 );

XH.write( OR132

| OR134 );

OR121.write( (P22 & (~((sc_logic)AI) & ~((sc_logic)CI) & EEI))

| (P13 & ~((sc_logic)EI))) ;
```

```
OR122.write((AI & BI & EI & II)
| (~((sc_logic)CI) & ~((sc_logic)DI) & ~((sc_logic)EI) & ~((sc_logic)II))
| (P31 & II));
OR123.write( (P31 & II)
| (P22 & BI & CI & EEI)
| (P13 & DI & EI & II)) ;
OR124.write( (P22 & AI & CI & EEI)
| (P13 & ~((sc_logic)EI))) ;
OR125.write((P13 & ~((sc_logic)EI))
| (~((sc_logic)CI) & ~((sc_logic)DI) & ~((sc_logic)EI) & ~((sc_logic)II))
| (~((sc_logic)AI) & ~((sc_logic)BI) & ~((sc_logic)EI) & ~((sc_logic)II))) ;
OR126.write( (P22 & ~((sc_logic)AI) & ~((sc_logic)CI) & EEI)
| (P13 & ~((sc_logic)II))) ;
OR127.write( (P13 & DI & EI & II)
| (P22 & ~((sc_logic)BI) & ~((sc_logic)CI) & EEI)) ;
XA.write( OR127 | OR121 | OR122 ) ;
XB.write( OR122 | OR123 | OR124 );
XC.write( OR121 | OR123 | OR125 );
XD.write( OR122 | OR124 | OR127 );
XE.write( OR125 | OR126 | OR127 );
output.notify(1.5,SC_NS);
}//while
} // end process DEC_6B_IN
void dec_8b10b::DEC5B(void){
wait(output);
```

```
while(true) {

wait(1);

if (RESET == SC_LOGIC_1){

AO.write( SC_LOGIC_0 ) ;

BO.write( SC_LOGIC_0 ) ;

CO.write( SC_LOGIC_0 ) ;

DO.write( SC_LOGIC_0 ) ;

EO.write( SC_LOGIC_0 ) ;

}else if( RBYTECLK.negedge()){

AO.write( XA ^ AI );

BO.write( XB ^ BI );

CO.write( XC ^ CI );

DO.write( XD ^ DI );

EO.write( XE ^ EI );

} //end if

}//while

} //end process DEC5B

void dec_8b10b::DEC3B ( void ){

wait(output);

while(true) {

wait(1);

if (RESET == SC_LOGIC_1){

FO.write(SC_LOGIC_0) ;

GO.write(SC_LOGIC_0) ;

HO.write(SC_LOGIC_0) ;
```

```
}else if( RBYTECLK.negedge()){

FO.write( XF ^ FI ) ;

GO.write( XG ^ GI ) ;

HO.write( XH ^ HI ) ;

} //end if

wait(SC_ZERO_TIME);

}//while

} //end process DEC3B
```

Testbench

```
#include "stimulus.h"

int sc_main (int argc, char* argv[]) {

sc_report_handler::set_actions("/IEEE_Std_1666/deprecated", SC_DO_NOTHING);

sc_clock clock("TSBYTECLK",0.5,0.5); //Master synchronous send byte clock

sc_signal<sc_lv<8> > input_v;

sc_signal<sc_lv<10> > output_v;

sc_signal<sc_lv<10> > dec_input_v;

sc_signal<sc_lv<8> > dec_output_v;

//instantaition stimulus

stimulus stim("STIM");

// Connect the Stimulus

stim.clock(clock);

stim.input_v(input_v);

stim.output_v(output_v);

stim.dec_input_v(dec_input_v);
```

```
stim.dec_output_v(dec_output_v);

//Open SystemC Trace files

sc_trace_file *EN = sc_create_vcd_trace_file("encoder_trace_file");

sc_trace(EN, clock, "clock");

sc_trace(EN,input_v,"8bit_Enc");

sc_trace(EN,output_v,"10bit_Enc");

sc_trace(EN,dec_input_v,"10bit_Dec");

sc_trace(EN,dec_output_v,"8bit_Dec");

sc_start(400);

sc_close_vcd_trace_file(EN);

return 0; //Terminate simulation

} //END MAIN
```

Appendix B : Wireless Channel Model

Node Model

File: Node.h

```
#include "systemc.h"

// Transmiter Module

SC_MODULE(transmitter) {

sc_in<bool> clock;

// output port

sc_port<sc_fifo_out_if<sc_bv<48> >,0 > tpackout;

// input port

sc_port<sc_fifo_in_if<sc_bv<48> >,0 > ack;

// Local variables

sc_bv <16> old_data[8];

sc_event e_transmitter;

sc_uint<64> i_fram_transmit_time;

sc_uint<64> ack_time_out[8];

sc_int<32> Transmiter_packet_count ;

sc_uint<3> N;
```

```
sc_uint<3> SNmin;

sc_uint<3> SNmax;

sc_uint<3> resend_couter;

// process

void snd_proc();

void snd_ack_proc();

// function

int RNG();

SC_CTOR(transmitter){

cout << "creating Transmitter name = " << name() << endl;

SC_THREAD(snd_proc);

sensitive << clock.pos();

dont_initialize();

SC_THREAD(snd_ack_proc);

sensitive << clock.neg();

dont_initialize();

Transmiter_packet_count = 1;

}

};

// Receiver Module

SC_MODULE(receiver) {

sc_in<bool> clock;

// input port

sc_port<sc_fifo_in_if<sc_bv<48> > > rpackin;

// output port
```

```
sc_port<sc_fifo_out_if<sc_bv<48> > > ack;

// Local variables

sc_bv<48> i_frame;

sc_bv<48> old_i_frame;

sc_bv<48> i_frame_ack;

sc_int<32> Receive_packet_count;

sc_uint<32> packet_error_count;

sc_uint<32> packet_successful_count;

double PE;

double throughput;

sc_uint<64> simTime;

sc_uint<3> RN;

sc_uint<3> SN;

// process

void receiver_proc();

SC_CTOR(receiver){

cout << "creating Receiver name = " << name() << endl;

SC_THREAD(receiver_proc);

sensitive << rpackin << clock.neg();

dont_initialize();

Receive_packet_count = 0;

packet_error_count = 0;

packet_successful_count = 0;

PE = 0.0;

throughput = 0.0;

simTime = 0;

}

};
```

File: Node.cpp

```
#include <string.h>

#include "node.h"

#include "sc_string.h"

#include <iomanip>

using namespace std;

void transmitter::snd_proc() {

int randno = 0;

sc_bv<48> i_frame;

sc_bv<8> address = "00000000";

sc_bv<8> control = "00000000";

sc_bv<16> i1 = "0000000000000000";

sc_bv<16> i2 = "0000000000000000";

sc_bv<16> checksum = "0000000000000000";

sc_bv<16> temp_data;

std::string get_node_name;

wait(10, SC_NS);

SNmin = 0;

SNmax = 0;

N = 7;

resend_couter = 0;

while(true){

wait();

randno = RNG();

wait(randno, SC_PS);
```

```
get_node_name = name();

temp_data = RNG();

if( SNmax < (SNmin + N)){

//Send next packet SNmax

if( resend_couter <= 0 ){

for (int count=0; count != tpackout.size(); count++) {

i_frame.range(47,40) = count;

i_frame.range(31,16) = temp_data;

i_frame.range(34,32) = SNmax;

i1 = i_frame.range(47,32);

i2 = i_frame.range(31,16);

// Checksum calculation

checksum = ~(i1 ^ i2);

i_frame.range(15,0) = checksum;

tpackout[count]->write(i_frame);

cout<<"@"<<sc_time_stamp()<<"Address"<<i_frame.range(47,40).to_int()<<endl;

cout<<"@"<<sc_time_stamp()<<"control"<<i_frame.range(34,32).to_uint()<<endl;

cout<<"@"<<sc_time_stamp()<<"information"<<i_frame.range(31,16).to_int()<<endl;

cout<<"@"<<sc_time_stamp()<<"checksum‘‘<< i_frame.range(15,0).to_int()<<endl;

} // end for loop

old_data[SNmax] = temp_data;

}else{

for (int count=0; count != tpackout.size(); count++) {

i_frame.range(47,40) = count;

i_frame.range(31,16) = old_data[SNmax];
```

```
i_frame.range(34,32) = SNmax;

i1 = i_frame.range(47,32);

i2 = i_frame.range(31,16);

checksum = ~(i1 ^ i2);

i_frame.range(15,0) = checksum;

tpackout[count]->write(i_frame);

}

cout <<"@"<<sc_time_stamp()<<"<><> i_fram RETRANSMITION <><>\n"<<endl;

resend_couter--;

}

i_fram_transmit_time = sc_time_stamp().value();

ack_time_out[SNmax] = i_fram_transmit_time + 150000;//300000;

cout<<" Transmitter send i_frame["<<SNmax<<" ] = "<<i_frame<<endl;

SNmax++;

}else{

wait(e_transmitter);

}

// }// end for

cout<<"number of packets transmitted="<< Transmiter_packet_count<<"\n"<<endl;

Transmiter_packet_count = Transmiter_packet_count + 1;

}// end while

} // end snd_proc

void transmitter::snd_ack_proc() {

sc_bv<48> i_frame;

sc_uint<3> RN;
```

```
sc_uint<3> R;

int address;

int rn=0;

int check_rn[8][2]={{0,0},{0,0},{0,0},{0,0},{0,0},{0,0},{0,0},{0,0}};

//sc_uint<3> old_SNmax = 0;

while(true){

wait();

for (int count=0; count <2; count++) {

if (ack[count] -> nb_read(i_frame)) {

RN = i_frame.range(34,32).to_uint();

cout<< " RN (ack Process) = "<< RN << endl;

rn = RN;

address = i_frame.range(47,40).to_uint();

// fill RN table

cout<<"***** rn = "<< rn << " ****** address = "<< address <<endl;

check_rn[rn][address]=1;

// printout the table

cout<<"check_rn["<<rn<<"]["<<address<<"]="<<check_rn[rn][address]<<endl;

for(int i=0;i<8;i++)

{

cout<<check_rn[i][0]<<" "<< check_rn[i][1]<<endl;

}

if(check_rn[rn][0]*check_rn[rn][1])

{

SNmin = RN;
```

```
R = RN;

cout<< " RN (inside IF) = "<< RN << endl;

check_rn[rn][0]=0;

check_rn[rn][1]=0;

}

}else if (ack_time_out[R] <= (sc_time_stamp().value())){

cout << "\n@" << sc_time_stamp()<< "<><><> TIME OUT <><><> "<<endl;

cout<<" ack_time_out-R["<<R<<"]="<< ack_time_out[R]<<endl;

SNmax = R;

resend_couter = 7;

}// elseif

}// end for

}// end while

} // end snd_ack_proc

////////////RECEIVER PROCESS////////////////////

void receiver::receiver_proc() {

std::string get_node_name;

sc_bv<16> i1 ="0000000000000000";

sc_bv<16> i2 ="0000000000000000";

sc_bv<16> checksum ="0000000000000000";

sc_bv<16> temp ="0000000000000000";

old_i_frame.range(39,32)="00000001";

RN = 0;

//create a file name

std::ostringstream file_name;
```

```
file_name << "ber" << name() <<".dat"<< std::ends;

std::string s = file_name.str();

ofstream myfile (s.c_str());

myfile <<"ST\t\tRPC\t\tPSC\t\tPEC\t\tTP\t\tPE"<<endl;

while(true) {

wait();

i_frame = rpackin->read(); // read input port

Receive_packet_count++;

simTime = sc_time_stamp().value();

cout<<" "<<endl;

cout<<sc_time_stamp()<<"address"<<i_frame.range(47,40).to_int()<<endl;

cout<<sc_time_stamp()<<"control"<<i_frame.range(34,32).to_uint()<<endl;

cout<<sc_time_stamp()<<"info"<<i_frame.range(31,16).to_int()<<endl;

cout<<sc_time_stamp()<<"checksum ‘‘<< i_frame.range(15,0).to_int()<<endl;

i1 = i_frame.range(47,32);// firest 16 bit section of i_frame

i2 = i_frame.range(31,16);// second 16 bit section of i_frame

checksum = i_frame.range(15,0);

// Checksum calculation

temp = ~((i1 ^ i2) ^ checksum);

if(temp == 0 ){

cout << "\n ************\n successful Transmition"<<endl;

cout << " ********************\n" << endl;

SN = i_frame.range(34,32).to_uint();

if (SN == RN ){

// Packet Accepted
```

```
packet_successful_count++;

RN++;

get_node_name = name();

if(get_node_name.compare("Receiver1"))

{ i_frame_ack.range(47,40) = "00000000"; }

else

{ i_frame_ack.range(47,40) = "00000001"; }

i_frame_ack.range(31,16) = "0000000000000000";

i_frame_ack.range(34,32) = RN;

ack -> write(i_frame_ack);

} else {

cout<<"\n Frame discarded in :  "<< name()<<" RN not equal SN "<<endl;

cout<<" Because SN = "<< SN <<" RN = "<< RN<<endl; }

} else {

packet_error_count++;

cout << "\n Transmition failer"<<endl;

cout << "\n" << endl;

// suspend til new i_frame arrive

wait(rpackin->data_written_event());

throughput = ( packet_successful_count.to_double() ) /

( Receive_packet_count.to_double() );

PE = ( packet_error_count.to_double() ) /

( Receive_packet_count.to_double() );

}//end else

if(Receive_packet_count>=200)
```

```
{sc_stop(); cout<<"Program terminated by number of packets"<<endl;}

}//while

myfile.close();

} // end of receiver_proc

int transmitter::RNG() {

int min =0;

int max =1000;

int randomNumber;

randomNumber = min + int(1.0*(max-min+1)*rand()/(RAND_MAX+1.0));

return randomNumber;

}
```

Channel Model

File: channel.h

```
#include "node.h"

#include "rng.h"

SC_MODULE(channel) {

sc_in<bool> clock;

//input ports

sc_port<sc_fifo_in_if<sc_bv<48> > > channel_in;

sc_port<sc_fifo_in_if<sc_bv<48> > > ack_in;

// output ports

sc_port<sc_fifo_out_if<sc_bv<48> > > channel_out;

sc_port<sc_fifo_out_if<sc_bv<48> > > ack_out;
```

```
// data frames process

void transfer();

// Acknowledgement fram process

void ack_process();

SC_CTOR(channel) {

cout << "creating Channel name = " << name() << endl;

SC_THREAD(transfer); // THREAD Process

sensitive << channel_in<< clock.pos();

dont_initialize();

SC_THREAD(ack_process); // THREAD Process

sensitive << ack_in<<clock.pos();

dont_initialize();

}

private:

MyRNG rng; // Random number class instantuation .

};
```

File: channel.cpp

```
#include "channel.h"

void channel::transfer() {

// local variables

sc_bv<48> i_frame;

sc_uint<64> next_error_position = 0;

double lamda = 0.0;

sc_uint<64> simTime = 0;
```

```
sc_uint<64> packet_counter =0;

lamda = rng.uniform(0.0001,0.01);

next_error_position = rng.exponential(lamda);

simTime = sc_time_stamp().value();

cout<<"\n"<<name()<<" initial error position"<<endl;

cout<<"next_error_position="<<next_error_position<<"Lamda="<<lamda<< endl;

while(true) {

wait();

i_frame = channel_in->read();

packet_counter++;

cout<<sc_time_stamp()<<"address="<<i_frame.range(47,40).to_int()<<endl;

cout<<sc_time_stamp()<<"control="<<i_frame.range(39,32).to_int()<<endl;

cout<<sc_time_stamp()<<"information="<<i_frame.range(31,16).to_int()<<endl;

cout<<sc_time_stamp()<<"checksum="<<i_frame.range(15,0).to_int()<<endl;

simTime = sc_time_stamp().value();

if (packet_counter >= next_error_position){

packet_counter =0;

i_frame = ~i_frame;

next_error_position = rng.exponential(lamda);

simTime = sc_time_stamp().value();

}

channel_out -> write( i_frame );

}//end while

} // Transfer process

// Acknowledgement process
```

```
void channel::ack_process() {

// local variables

sc_bv<48> i_frame;

while(true) {

wait();

i_frame = ack_in->read();

cout<<"Acknowledgement in Channel:"<<name()<<"received"<<i_frame.range(47,35)<<

"["<<i_frame.range(34,32)<<" ]"<<i_frame.range(31,0)<<endl;

ack_out -> write( i_frame );

}//end while

} // Acknowledgement process
```

File: main.cpp

```
#include "channel.h"

int sc_main (int argc, char* argv[]) {

sc_report_handler::set_actions("/IEEE_Std_1666/deprecated", SC_DO_NOTHING);

sc_clock Tclock ("TCLOCK",100,0.5);

sc_clock Rclock ("RCLOCK",100,0.5);

sc_clock channel_clock ("CHANNEL_CLOCK",100,0.5);

// modules instantiations

unsigned NTransmitters = 1;

unsigned MReceivers = 2;

unsigned MChannels = 2;

transmitter *Transmitter[NTransmitters];

receiver *Receiver[MReceivers];
```

```
channel *Channel[MChannels];

sc_fifo<sc_bv<48> > *TCh_fifo[MReceivers];

sc_fifo<sc_bv<48> > *ChR_fifo[MReceivers];

sc_fifo<sc_bv<48> > *ChTAck_fifo[MReceivers];

sc_fifo<sc_bv<48> > *RChAck_fifo[MReceivers];

for (unsigned i=0; i < NTransmitters; i++) {

//create a module

std::ostringstream trans_name;

trans_name << "Transmitter" << i << std::ends;

std::string s = trans_name.str();

Transmitter[i] = new transmitter(s.c_str());

}

// Generate M Receiver, M Channel, M TCh_fifo and M ChR fifo

for (unsigned i=0; i < MReceivers; i++) {

//create a receivers modules

std::ostringstream recev_name;

recev_name << "Receiver" << i << std::ends;

std::string s1 = recev_name.str();

Receiver[i] = new receiver(s1.c_str());

// create Channels modules

std::ostringstream ch_name;

// create fifos

ch_name << "Channel" << i << std::ends;

std::string s2 = ch_name.str();

Channel[i] = new channel(s2.c_str());
```

```
// create TCh_fifos
TCh_fifo[i] = new sc_fifo<sc_bv<48> >;
// create ChR_fifos
ChR_fifo[i] = new sc_fifo<sc_bv<48> >;
// create TChAck_fifos
ChTAck_fifo[i] = new sc_fifo<sc_bv<48> >;
// create ChRAck_fifos
RChAck_fifo[i] = new sc_fifo<sc_bv<48> >;
}
// this for loop to bind the Transmitter to Channels trough TCh_fifo
Transmitter[0]->clock(Tclock);
for (unsigned i=0; i< MChannels; i++){
//bind fifo to module port
Transmitter[0]->tpackout(*TCh_fifo[i]);
cout<<"\nTransmitter"<<0<<"->Tpackout(*TCh_fifo["<<i<<"])"<<endl;
Transmitter[0]->ack(*ChTAck_fifo[i]);
cout<<"\nTransmitter"<<0<<"->ack(*ChAck_fifo["<<i<<"])"<<endl;
}
//to bind the Receivers to ChR_fifos and Receivers to RComp_fifos
for (unsigned i=0; i< MReceivers; i++){
Receiver[i]->clock(Rclock);
//bind fifo to module port
Receiver[i]->rpackin(*ChR_fifo[i]);
cout<<"\nReceiver"<<i<<"->rpackin(*ChR_fifo["<<i<<"])"<<endl;
Receiver[i]->ack(*RChAck_fifo[i]);
```

```
cout<<"\nReceiver"<<i<<"->ack(*RChAck_fifo["<<i<<"])"<<endl;

}

// this for loop to bind the Channels to ChR_fifos and TCh_fifos

for (unsigned i=0; i< MChannels; i++){

Channel[i]->clock(channel_clock);

//bind fifo to module port

Channel[i]->channel_in(*TCh_fifo[i]);

cout<<"\nChannel"<<i<<"->channel_in(*TCh_fifo["<<i<<"])"<<endl;

Channel[i]->channel_out(*ChR_fifo[i]);

cout<<"\nChannel"<<i<<"->channel_out(**ChR_fifo["<<i<<"])"<<endl;

Channel[i]->ack_out(*ChTAck_fifo[i]);

cout<<"\nChannel"<<i<<"->ack_out(*ChTAck_fifo["<<i<<"])"<<endl;

Channel[i]->ack_in(*RChAck_fifo[i]);

cout<<"\nChannel"<<i<<"->ack_in(**ChR_fifo["<<i<<"])"<<endl;

}

sc_start();

return(0);

}
```

Appendix C : Flocking Behaviour System

File: Node.h

```
#ifndef NODE_H

#define NODE_H

#include "systemc.h"

#include <iostream.h>

#include <iomanip>

#include "channel.h"

#include "uniform.h"

#include "node_average.h"

// N is the Number of particles

#define RATE 1

SC_MODULE(node) {

// input port

sc_port<sc_fifo_in_if<packet_type> > input;

sc_port<sc_fifo_in_if<packet_type> > input2;

// output port

sc_port<sc_fifo_out_if<packet_type> > output;

sc_port<sc_fifo_out_if<packet_type> > output2;
```

```
sc_port<sc_fifo_out_if<double> > average_port;

sc_port<sc_fifo_out_if<int> > simtime_port;

// process

void snd_proc();

void receiver_proc();

// variables and functions

int relposx;

int relposy;

double xspeed;

double yspeed;

double xspeedold;

double yspeedold;

double accel;

double kd;

bool leader;

bool exit;

double total_speed;

void estimatePosition(double *accx, double *accy);

double x;

double y;

double accx;

double accy;

void setLeader(double xposL,double yposL,double xvel,double yvel);

//void actions(void);

double dx;
```

```
double dy;

double d_total;

double olddx;

double olddy;

double xacceleration;

double yacceleration;

// node data variables

std::string get_node_name;

packet_type info;

packet_type read_input_port[2];

sc_uint<64> simTime;

double max_delay;

int packet_number1;

int packet_number2;

randomNumber test;

SC_HAS_PROCESS(node);

node(sc_module_name nm, packet_type nodeData) :  sc_module(nm)

{

cout << "Constructor :  creating particle --> " << name() << endl;

info.id = nodeData.id;

info.relposx = nodeData.relposx;

info.relposy = nodeData.relposy;

info.xpos = nodeData.xpos;

info.ypos = nodeData.ypos;

// Initial values
```

```
cout << "Initial Values for node --> " << name() << endl;

cout<<" id = "<< nodeData.id <<endl;

cout<<" relposx = "<< nodeData.relposx <<endl;

cout<<" relposy = "<< nodeData.relposy <<endl;

cout<<" xpos = "<< nodeData.xpos <<endl;

cout<<" ypos = "<< nodeData.ypos <<endl;

xspeed=0.0;

yspeed=0.0;

xspeedold=0.0;

yspeedold=0.0;

accel=0.03;

kd=0.1;

leader=false;

exit=false;

SC_THREAD(snd_proc);

SC_THREAD(receiver_proc);

}

private:  sc_uint<64> count_step;

sc_uint<64> cycle_number;

};

#endif
```

File: Node.cpp

```
#include "node.h"

using namespace std;

// SEND PROCESS

void node::snd_proc() {

std::string node_name;

node_name=name();

double starting_delay=test.uniform(0.0,RATE);

cycle_number=1;

count_step=RATE;

int convert;

convert=static_cast<int> (starting_delay);

std::ostringstream file_name55;

file_name55 << "CHECK-RATE_"<< name() <<".dat"<< std::ends;

std::string s55 = file_name55.str();

ofstream myfile55 (s55.c_str());

while(true) {

wait(1,SC_MS);

//TO CONTROL TRANSMISSION RATE and ADD INITIAL DELAY

if((convert+count_step)%RATE==0)

{

myfile55 << sc_time_stamp() << setiosflags (ios::left) << setw(8) <<

" TRANSMISSION @ cycle-number ” << cycle_number++ << "count_step = "

<< count_step << endl;

output->nb_write(info);
```

```
output2->nb_write(info);

}

count_step++;

}// end while

}// end process

// RECEIVER PROCESS

void node::receiver_proc() {

double tempx;

double tempy;

std::string node_name;

node_name=name();

dx=0.0;

dy=0.0;

std::ostringstream file_name;

file_name << "practicle_behaviour_"<< name() <<".dat"<< std::ends;

std::string s = file_name.str();

ofstream myfile (s.c_str());

std::ostringstream file_name1;

file_name1 << "practicle_dynamics_"<< name() <<".dat"<< std::ends;

std::string s1 = file_name1.str();

ofstream myfile1 (s1.c_str());

std::ostringstream file_name2;

file_name2 << "practicle_converge_"<< name() <<".dat"<< std::ends;

std::string s2 = file_name2.str();

ofstream myfile2 (s2.c_str());
```

```
std::ostringstream file_name5;

file_name5 << "temp_"<< name() <<".dat"<< std::ends;

std::string s5 = file_name5.str();

ofstream myfile5 (s5.c_str());

while(true) {

wait(input->data_written_event()|input2->data_written_event());

simTime = sc_time_stamp().value()/1000000000;

olddx=dx;

olddy=dy;

input ->nb_read(read_input_port[0]); // read value

input2 ->nb_read(read_input_port[1]); // read value

// CALL ESTIMATE_POSITION Function

estimatePosition(&dx, &dy);

if (leader)

{

dx=0;

dy=0;

}// end if

if (!leader)

{

//#move towards my goal; so move speed according to error:

xacceleration=dx*accel+(dx-olddx)*kd;

if (xacceleration>0.2)

xacceleration=0.2;

if (xacceleration<-0.2)
```

```
xacceleration=-0.2;

yacceleration=dy*accel+(dy-olddy)*kd;

if (yacceleration>0.2)

yacceleration=0.2;

if (yacceleration<-0.2)

yacceleration=-0.2;

xspeed+=xacceleration;

yspeed+=yacceleration;

//#should probably saturate acceleration

if (xspeed>1.)xspeed=1.0;

if (xspeed<-1.)xspeed=-1.;

if (yspeed>1.)yspeed=1.;

if (yspeed<-1.)yspeed=-1.;

}// end if

total_speed= sqrt(pow(xspeed,2)+pow(yspeed,2));

//#Now, calculate new position for next time we are called;

//#So at end of this clock cycle our position would be:

info.xpos+=xspeed;

info.ypos+=yspeed;

cout<<sc_time_stamp() <<" LAST OUTPUT:" << " xpos = "<< info.xpos << ",ypos=
"<< info.ypos << endl;

tempx= abs(info.xpos-static_cast<double>(info.relposx));

tempy= abs(info.ypos-static_cast<double>(info.relposy));

myfile5<< setiosflags(ios::left) << setw(8) << tempx << "\t" << setw(8) << tempy
<< "\t"<< setw(8) << info.relposx << "\t" << setw(8) << info.xpos << "\t" <<
setw(8) << info.relposy << "\t" << setw(8) << info.ypos << endl;
```

```
// writing output data to file

// 1) Create a file for xpos and ypos

myfile << setiosflags(ios::left) << setw(8) << info.xpos << "\t" << setw(8) <<
info.ypos << endl;

// 2) create a file for dx and dy

myfile1 << setiosflags(ios::left) << setw(8) << simTime << "\t" << setw(8) <<
dx << endl;

d_total= sqrt(pow(dx,2)+pow(dy,2));

// create a file for dx and dy

myfile2<< setiosflags(ios::left) << setw(8) << simTime << "\t" << setw(8) <<
d_total << endl;

if(!node_name.compare(name())&& node_name.compare("Node0") )

{

if(simTime > 10)

{

average_port->nb_write(d_total);

simtime_port->nb_write(simTime);

}

}// end outer if

}// end while

}// end process

//*************************

// function estimatePosition

//*************************

void node::estimatePosition(double *accx, double *accy)

{
```

```
*accx=0.0;

*accy=0.0;

//Estimates my *relative* position according to my links

// We have to link this variable with number of nodes

int links =2;

for(int adove =0; adove<links; adove++)

{

x=info.xpos-read_input_port[adove].xpos; ///relposx);

y=info.ypos-read_input_port[adove].ypos; //relposy);

*accx+=-read_input_port[adove].relposx+info.relposx-x;

*accy+=-read_input_port[adove].relposy+info.relposy-y;

}

*accx/=static_cast<float>(links);

*accy/=static_cast<float>(links);

}// end estimatePosition function

//********************

// function setLeader

//********************

void node::setLeader(double xposL,double yposL,double xvel,double yvel)

{

info.xpos=xposL;

info.ypos=yposL;

xvel=xvel;

yvel=yvel;

leader=true;

}// end setLeader function
```

File : channel.h

```
#include "systemc.h"

#include "packet_type.h"

SC_MODULE(channel) {

// input port

sc_port< sc_fifo_in_if<packet_type> > channel_input;

// output port

sc_port< sc_fifo_out_if<packet_type> > channel_output;

// process1

void p1();

// variables

packet_type get_data1;

packet_type lost_packet;

SC_CTOR(channel)

{

cout << "creating Channel name = " << name() << endl;

SC_THREAD(p1); // THREAD Process

}

};
```

File : channel.cpp

```
#include "channel.h"

void channel::p1() {

while(true)

{
```

```
wait(channel_input->data_written_event());

get_data1 = channel_input->read();

cout << sc_time_stamp() << "Channel get the value from transmitter:"

<< get_data1 << endl;

channel_output-> write(get_data1);

}//end while

}// end process
```

File : converge.h

```
#ifndef NODE_AVERAGE_H

#define NODE_AVERAGE_H

#include "systemc.h"

#include <iostream.h>

#include <iomanip>

// N is the Number of particles

#define N 10

SC_MODULE(node_average) {

/// input port

sc_port<sc_fifo_in_if <double> ,0> one_input_port[N];

sc_port<sc_fifo_in_if <int> ,0> simtime_input_port[N];

double read_error[N];

int simtime_converge[N];

double total_error;

int total_simtime;

void average_proc();
```

```
SC_CTOR(node_average) {

total_error=0.0;

total_simtime=0;

SC_THREAD(average_proc);

}

};

#endif
```

File : converge.cpp

```
#include "node_average.h"

using namespace std;

void node_average::average_proc() {

std::ostringstream file_final;

file_final<<"error_final_" << name() <<".dat"<< std::ends;

std::string s99 = file_final.str();

ofstream myfile99 (s99.c_str());

int noo=1;

double CAA,CAA1;

CAA=0.0;

while(true) {

wait(1,SC_MS);

for(int i = 1; i<N;i++)

{

one_input_port[i]->read(read_error[i]);

simtime_input_port[i]->read(simtime_converge[i]);
```

```
total_error=total_error + read_error[i];

total_simtime=total_simtime + simtime_converge[i];

}

total_error = total_error/N;

total_simtime = total_simtime/N;

CAA1=CAA+(total_error-CAA)/noo;

myfile99 << setiosflags(ios::left) << setw(8) << total_simtime << "\t"

<< setw(8) << CAA1 << endl;

if(CAA1<0.005)

{

cout << "Converging Time=" << total_simtime << "@ Average Error Value="

<< total_error << endl;

cout << name() << "****** Simulation Stopped by US" << endl;

sc_stop();

}

CAA=CAA1;

noo++;

}// end while

}// end process
```

File :main.cpp

```
#include "node.h"

#include "systemc.h"

using namespace std;

int sc_main (int argc, char* argv[])
```

```
{
sc_report_handler::set_actions("/IEEE_Std_1666/deprecated",SC_DO_NOTHING);
// create array of FIFOs
sc_fifo<packet_type> fifo_chan[4*N];
sc_fifo<double> fifo_chan3[N];
sc_fifo<int> fifo_chan4[N];
// create object from datavxy class
packet_type startingData;
// particles instantiations
unsigned adove_N = N;
node *adove[adove_N];
channel *ch[N];
channel *ch2[N];
node_average *average;
std::ostringstream file_name_main;
file_name_main << "relative"<<".dat"<< std::ends;
std::string s_main = file_name_main.str();
ofstream myfile_main (s_main.c_str());
startingData.relposx = 0;
startingData.relposy = 0;
for (unsigned i=0; i < adove_N; i++)
{
std::ostringstream particle_name;
particle_name << "Node" << i << std::ends;
std::string s = particle_name.str();
```

```
startingData.id = i;

startingData.xpos = 0.0;

startingData.ypos = 0.0;

if(i==0)

{startingData.relposx = 0;

startingData.relposy = 0;}

if(i==1)

{startingData.relposx = 1;

startingData.relposy = 0;}

if(i==2)

{startingData.relposx = 2;

startingData.relposy = 0;}

if(i==3)

{startingData.relposx = 3;

startingData.relposy = 0;}

if(i==4)

{startingData.relposx = 4;

startingData.relposy = 0;}

if(i==5)

{startingData.relposx = 5;

startingData.relposy = 0;}

if(i==6)

{startingData.relposx = 5;

startingData.relposy = 1;}

if(i==7)
```

```
{startingData.relposx = 5;
startingData.relposy = 2;}
if(i==8)
{startingData.relposx = 5;
startingData.relposy = 3;}
if(i==9)
{startingData.relposx = 5;
startingData.relposy = 4;}
if(i==10)
{startingData.relposx = 5;
startingData.relposy = 5;}
if(i==11)
{startingData.relposx = 4;
startingData.relposy = 5;}
if(i==12)
{startingData.relposx = 3;
startingData.relposy = 5;}
if(i==13)
{startingData.relposx = 2;
startingData.relposy = 5;}
if(i==14)
{startingData.relposx = 1;
startingData.relposy = 5;}
if(i==15)
{startingData.relposx = 0;
```

```
startingData.relposy = 5;}

if(i==16)

{startingData.relposx = 0;

startingData.relposy = 4;}

if(i==17)

{startingData.relposx = 0;

startingData.relposy = 3;}

if(i==18)

{startingData.relposx = 0;

startingData.relposy = 2;}

if(i==19)

{startingData.relposx = 0;

startingData.relposy = 1;}

myfile_main << setiosflags(ios::left) <<setw(8) << startingData.relposx << "\t"
<< setw(8)<< startingData.relposy << endl;

adove[i] = new node(s.c_str(),startingData);

}

average = new node_average("node_average");

// Creating channel

cout<<"\n*************** CREATING CHANNEL***********\n"<<endl;

for (unsigned i=0; i < N; i++)

{

//create particles (or nodes)

std::ostringstream ch_name;

ch_name << "Channel" << i << std::ends;
```

```
std::string s1 = ch_name.str();

ch[i] = new channel(s1.c_str());

}

for (unsigned i=0; i < N; i++)

{

//create particles (or nodes)

std::ostringstream ch_name2;

ch_name2 << "Channel2" << i << std::ends;

std::string s2 = ch_name2.str();

ch2[i] = new channel(s2.c_str());

}

//Binding

for (unsigned i=0; i < N; i++)

{

adove[i]->average_port(fifo_chan3[i]);

average->one_input_port[i](fifo_chan3[i]);

adove[i]->simtime_port(fifo_chan4[i]);

average->simtime_input_port[i](fifo_chan4[i]);

}

//port Node:  input

for (unsigned i=0; i < adove_N; i++)

{

adove[i]->input(fifo_chan[4*i+1]);

}

//port Node:  input2
```

```
for (unsigned i=1; i <= adove_N; i++)

{

adove[i%adove_N]->input2(fifo_chan[4*i-2]);

}

//port Node:  output

for (unsigned i=0; i < adove_N; i++)

{

adove[i]->output(fifo_chan[4*i]);

}

//port Node:  output2

for (unsigned i=1; i <= adove_N; i++)

{

adove[i%adove_N]->output2(fifo_chan[4*i-1]);

}

// Binding Channels

//port ch:  input

for (unsigned i=0; i < adove_N; i++)

{

ch[i]->channel_input(fifo_chan[4*i]);

}

//port ch:  output

for (unsigned i=1; i <= adove_N; i++)

{

ch[i-1]->channel_output(fifo_chan[4*i-2]);

}
```

```
//port ch2:  input
for (unsigned i=1; i <= adove_N; i++)
{
ch2[i-1]->channel_input(fifo_chan[4*i-1]);
}
//port ch2:  output
for (unsigned i=0; i < adove_N; i++)
{
ch2[i]->channel_output(fifo_chan[4*i+1]);
}
// select leader
adove[0]->setLeader(0.,0.,0.,0.);
for (unsigned i=0; i < adove_N; i++)
{
cout<<"\nadove["<< i <<"] - LEADER ---> "<<adove[i]->leader<<endl;
}
cout<< " "<<endl;
sc_start();
return(0);
}
```

Bibliography

[1] L. Cai, S. Verma, and D. Gajski, "Comparison of specc and systemc languages for system design," tech. rep., Center for Embedded Computer Systems, University of California, Irvine, 2003.

[2] M. C. Newsletter, "C2000 dsc tips and tricks," tech. rep., Texas Instruments, 2004.

[3] R. Domer, D. Gajski, and A. Gerstlauer, "Specc methodology for high-level modelling," *IEEE Electronic Design Processes Workshop EDP 2002*, 2002.

[4] D. Gajski, J. Zhu, R. Domer, A. Gerstlauer, and S. Zhao, *Spec C:Specification Language and Methodology*. 2000.

[5] IEEE_Computer_Society, "Ieee standard systemc language reference manual," *IEEE Std 1666-2005*, pp. 1–423, Mar. 2006.

[6] OSCI, *SystemC User's Guide*. Open SystemC Initiative, 2002.

[7] T. Groetker, S. Liao, G. Martin, and S. Swan, *System Design with SystemC*. 2002.

[8] D. Black and J. Donovan, *SystemC: From the Ground-up*. Kluwer Academic Publishers, first ed., 2004.

[9] SystemC-AMS, "http://www.systemc-ams.org/," *Accessed on 27/05/2008*.

[10] A. Vachoux, C. Grimm, and K. Einwick, "Analog and mixed signal modelling with systemc-ams," *IEEE*, 2003.

[11] C. Haubelt, J. Falk, J. Keinert, T. Schlichter, M. Streubuhr, A. Deyhle, A. Hadert, and J. Teich, "A systemc-based design methodology for digital signal processing systems," *EURASIP Journal on Embedded Systems*, vol. 2007, no. 1, 2007.

[12] F. Stefanni, D. Quaglia, and F. Fummi, "Systemc simulation of networked embedded systems," *Springer Netherlands*, vol. 36, pp. 201–211, May 2009.

[13] A. Ghosh, S. Tjiang, and R. Chandra, "System modeling with systemc," in *ASIC*, 2001.

[14] L. Cai and D. Gajski, "Transaction level modelling in system level design," *CECS Technical Report 03-10*, 2003.

[15] N. Bombieri, F. Fummi, and D. Quaglia, "System/network design-space exploration based on tlm for networked embedded systems," *ACM Transactions on Embedded Computing Systems*, vol. 9, Mar. 2010.

[16] DARPA and NSF, "The network simulator ns2."

[17] "Opnet - network simulation."

[18] D. Bourg and G. Seemann, *AI for Game Developers*. July 2004.

[19] I. Aref, N. Ahmed, F. Rodriguez-Salazar, and K. Elgaid, "Measuring and optimising convergence and stability in terms of system construction in systemc," *17th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS-2010)*, March 2010.

[20] I. Aref, N. Ahmed, F. Rodriguez-Salazar, and K. Elgaid, "Modeling of flocking behaviour system in systemc," *Sixth Advanced International Conference on Telecommuincations (AICT-2010)*, May 2010.

[21] I. Aref, N. Ahmed, F. Rodríguez-Salazar, and K. Elgaid, "Wireless extension into existing systemc design methodology," *The 2nd International Conference on Computer Engineering and Technology (ICCET 2010)*, April 2010.

[22] B. A. Forouzan, *Data Communications and Networking*. McGraw-Hill Science/Engineering/Math, forth ed., 2006.

[23] W. Yue and Y. Matsumoto, *Performance Analysis of Multi-Channel and Multi-Traffic on Wireless Communication Networks*. Kluwer Academic Publishers, first ed., 2002.

[24] S. Glisic and P. Leppänen, *Wireless Communications: TDMA versus CDMA*. Kluwer Academic Publishers, 1997.

[25] R. Yuce, P. C. Ng, C. Lee, J. Khan, and W. Liu, "A wireless medical monitoring over a heterogeneous sensor network," *Engineering in Medicine and Biology Society (EMBS-2007)*, 2007.

[26] J. Bhasker, *A SystemC Primer*. Star Galaxy Publishing, 2002.

[27] A. Sangiovanni-Vincentelli, "Is a unified methodology for system-level design possible?," *IEEE Design and Test of Computers*, 2008.

[28] Synopsys, *Describing Synthesizable RTL in SystemC*. 2002.

[29] W. Muller, W. Rosenstiel, and J. Ruf, *SystemC Methodologies and Applications*. Mar. 2003.

[30] OSCI, "Open systemc initiative (osci)."

[31] S. Swan, "An introduction to system level modeling in systemc 2.0," *Open SystemC Initiative (OSCI)*, May 2001.

[32] J. Gipper, "Systemc: the soc system-level modeling language," *Embedded Computing Design*, 2007.

[33] P. Panda, "Systemc - a modeling platform supporting multiple design abstractions," *The 14th International Symposium on System Synthesis*, 2001.

[34] J. DeGroat, A. Raman, and B. Younis, "A design project for system design with systemc," *Proceedings of the IEEE International Conference on Microelectronic Systems Education (MSE)*, 2003.

[35] D. Maliniak, "Systemc bridges the gap," *OSCI/OCP-IP Special Report - A Penton Publication*, Jan. 2005.

[36] The-MathWorks, "Matlab-the language of technical computing," 1996.

[37] The-MathWorks, "Simulink-simulation and model-based design," 1996.

[38] Celoxica, *Handle-C Language Reference Manual*, 2005.

[39] K. Ramamritham, K. Arya, and G. Fohler, "System software for embedded applications," *the 17th International Conference on VLSI Design (VLSID-04)*, 2004.

[40] L. Franzens, "Systemc for embedded system design," *University of Innsbruck, Institue for Informatics*, 2006.

[41] C. Widtmann, *High-level System Modeling with SystemC and TLM*. VDM Verlag, 2009.

[42] G. Domer and P. Gajski, *System Design: A Practical Guide with SpecC*. Kluwer Academic Publishers, 2001.

[43] R. Walstrom, J. Schneider, and D. Rover, "Teaching system-level design using specc and systemc," *Proceedings of the IEEE International Conference on Microelectronic Systems Education (MSE)*, 2005.

[44] L. Cai, D. Gajski, M. Olivares, and P. Kritzinger, "C/c++ based system design flow using specc, vcc and systemc," tech. rep., Center for Embedded Computer Systems, University of California, Irvine, 2002.

[45] L. Cai and D. Gajski, "Transaction level modeling: An overview," *CODES+ISSS*, 2003.

[46] F. Ghenassia, *Transaction Level Modeling with SystemC - TLM Concepts and Applications for Embedded Systems*. Springer, 2005.

[47] B. Jonsson, "A jpeg encoder in systemc," Master's thesis, Lulea University of Technology, Department of Computer Science and Electrical Engineering, Division of EISLAB, 2005.

[48] R. Domer, A. Gerstlauer, P. Kritzinger, and M. Olivarez, "The specc system-level design language and methodology, part 2," *Embedded Systems Conference San Francisco*, 2002.

[49] D. Gajski and F. Vahid, "Specfication design of embedded hardware-software systems," *IEEE Design and Test of Computers*, 1995.

[50] M. Pellmann, *Transaction Level Modelling Using SystemC*. VDM Verlag, 2008.

[51] J. Bjornsen and T. Ytterdal, "Behavioral modeling and simulation of high-speed analog-to-digital converters using systemc," *IEEE Circuits and Systems, ISCAS '03*, vol. 3, pp. III–906–III–909, May 2003.

[52] N. Calazans, E. Moreno, F. Hessel, V. Rosa, F. Moraes, and E. Carara, "From vhdl register transfer level to systemc transaction level modeling: a comparative case study," *IEEE International Symposium on Circuits and Systems*, 2003.

[53] S. Suresh, "System level design of a turbo decoder for communication systems," Master's thesis, Faculty of North Carolina State University, Electrical Engineering, 2005.

[54] R. Walstrom, "System-level design refinement using systemc," Master's thesis, Iowa State University, Ames, Iowa, 2005.

[55] Y. Qu, *System-level design and configuration management for run-time reconfigurable devices*. PhD thesis, Department of Information Technology, Tampere University of Technology, Nov. 2007.

[56] Y. Qu, K. Tiensyrja, and J. Soininen, "Systemc-based design methodology for reconfigurable system-on-chip," *8th Euromicro conference on Digital System Design (DSD)*, 2005.

[57] A. Vachoux, C. Grimm, and K. Einwick, "Towards analog and mixed-signal soc design with systemc-ams," *The second IEEE International Workshop on Electronic Design, Test and Applications (DELTA)*, 2004.

[58] A. Vachoux, C. Grimm, and K. Einwick, "Systemc-ams requirements, design objectives and rationale," *Design, Automation and Test in Europe, DATE03*, 2003.

[59] A. Vachoux, C. Grimm, and K. Einwick, "Extending systemc to support mixed discrete-continuous system modeling and simulation," *IEEE*, 2005.

[60] "Requirements specification for systemc analog mixed signal (ams) extensions," tech. rep., Open SystemC Initiative (OSCI), Dec. 2008.

[61] S. Orcioni, M. Ballicchia, G. Biagetti, R. Aparo, and M. Conti, "System level modelling of rf ic in systemc-wms," *EURASIP Journal on Embedded Systems*, no. 371768, 2008.

[62] G. Biagetti, M. Conti, and S. Orcioni, "Systemc-wms - http://www.deit.univpm.it/systemc-wms/," Sept. 2005.

[63] F. Fummi, S. Martini, and G. Perbellini, "Heterogeneous co-simulation of networked embedded systems," *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*, vol. 3, 2004.

[64] N. Drago, F. Fummi, and M. Poncino, "Modeling network embedded systems with ns-2 and systemc," *1st IEEE International Conference on Circuits and Systems for Communications (ICCSC)*, 2002.

[65] F. Fummi, P. Gallo, S. Martini, G. Perbellini, M. Poncino, and F. Ricciato, "A timing-accurate modeling and simulation environment for networked embedded systems," *Annual ACM IEEE Design Automation Conference*, pp. 42–47, 2003.

[66] F. Fummi, D. Quaglia, F. Ricciato, and M. Turolla, "Modeling and simulation of mobile gateways interacting with wireless sensor networks," *Proceedings of the conference on Design, automation and test in Europe*, pp. 106–111, Mar. 2006.

[67] N. Bombieri, F. Fummi, and D. Quaglia, "System/network design-space exploration based on tlm for networked embedded systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 9, Mar. 2010.

[68] E. Alessio, F. Fummi, D. Quaglia, and M. Turolla, "Modeling and simulation alternatives for the design of networked embedded systems," *Proceedings of the conference on Design, automation and test in Europe*, pp. 1030–1035, Apr. 2007.

[69] F. Fummi, D. Quaglia, and F. Stefanni, "A systemc-based framework for modeling and simulation of networked embedded systems," *Forum on Specfication, Verification and Design Languages*, pp. 49–54, Sept. 2008.

[70] C. Smith and C. Gervelis, *Wireless network performance handbook*. McGraw-Hill Network Engineering, 2003.

[71] M. Schiff, *Introduction to Communication Systems Simulation*. Artech House, 2006.

[72] R. Freeman, *Fundamentals of Telecommunications*. John Wiley and Sons, 1999.

[73] I. Glover and P. Grant, *Digital Communication*. Prentice Hall, 2000.

[74] D. Tse and P. Viswanath, *Fundamentals of Wireless Communication*. 2005.

[75] A. Arsal, "A study on wireless channel models: Simulation of fading, shadowing and further applications," Master's thesis, School of Engineering and Sciences of Izmir Institute of Technology, Aug. 2008.

[76] V. Kuhn, *Wireless Communications over MIMO Channels*. 2006.

[77] V. Lau and Y. Kwok, *Channel-Adaptive Technologies and Cross-Layer Designs for Wireless Systems with Multiple Antennas*. John Wiley and Sons, 2006.

[78] D. Johnson, "Channel models," tech. rep., Connexions Project, June 2009.

[79] R. Jain, "Channel models - a tutorials," tech. rep., Feb. 2007.

[80] M. Lindhe, "On communication and flocking in multi-robot systems," Master's thesis, KTH School of Electrical Engineering-Stockholm, 2007.

[81] H. Harada and R. Prasad, *Simulation and Software Radio for Mobile Communications*. Universal personal communications, 2002.

[82] D. Matolak, "Wireless channel characterization in the 5 ghz microwave landing system extension band for airport surface areas," tech. rep., Ohio University, Athens, Ohio, May 2007.

[83] L. Ahlin, J. Zander, and B. Slimane, *Principles of Wireless Communications*. Studentlitteratur AB, Aug. 2006.

[84] C. Haslett, *Essentials of Radio Wave Propagation*. Cambridge University Press, 2008.

[85] A. Garrido and D. Romera, "Radio wave propagation," tech. rep., Linköping University, Linköping, Sweden, 2007.

[86] "Communications-electronics fundamentals: Wave propagation, transmission lines, and antennas," tech. rep., Headquarters Department of the Army, July 2004.

[87] A. Goldsmith, *Wireless Communications*. Cambridge University Press, 2005.

[88] B. Sklar, *Digital Communications Fundamentals and Applications*. Prentice Hall, 2003.

[89] F. Halsall, *Data Communication, Computer Networks and Open Systems*. Addison-Wesley, 1995.

[90] A. Kumar, D. Manjunath, and J. Kuri, *Communication Networking An Analytical Approach*. Elsevier, 2004.

[91] H. Bidgoli, *The Handbook of Computer Networks*. John Wiley and Sons, 2008.

[92] M. Engels, *Wireless OFDM Systems-How to make them work*. Kluwer Academic Publishers, 2002.

[93] M. Jeruchim, P. Balaban, and K. Shanmugan, *Simulation of Communication Systems - Modeling, Methodology and Techniques*. Kluwer Academic Publishers, 2002.

[94] M. Simon and M. Alouini, *Digital Communications over Fading Channels*. Wiley Interscience, 2005.

[95] M. Ghavami, L. Michael, and R. Kohno, *Ultra Wideband - signals and systems in communication engineering*. John Wiley and Sons, 2004.

[96] H. Schulze and C. Luders, *Theory and Applications of OFDM and CDMA Wideband Wireless Communications*. John Wiley and Sons, 2005.

[97] M. Patzold, *Mobile Fading Channels*. John Wiley and Sons, 2002.

[98] J. Irvine and D. Harle, *Data Communications and Networks*. John Wiley and Sons, 2002.

[99] A. Ahmad, *Data Communication Principles For Fixed and Wireless Networks*. first ed., Jan. 2003.

[100] Y. Fan and Z. Zilic, "A novel scheme of implemening high speed awgn communication channel emulators in fpga," *IEEE-Proceedings of the 2004 International Symposium on Circuits and Systems ISCAS*, vol. 2, pp. 877–880, May 2004.

[101] F. Xiong, *Digital Modulation Techniques*. Artech House, 2000.

[102] H. Harada and R. Prasad, *Simulation and Software Radio for mobile communications*. second ed., May 2001.

[103] J. Zyren and A. Petrick, "Tutorial on basic link budget analysis," tech. rep., Intersil Corporation, 1998.

[104] N. Travers, "Simulated flocking behavior," tech. rep., Department of Mathematics, UC Davis, 2007.

[105] C. Hartman and B. Benes, "Autonomous biods," *Journal of Visualization and Computer Animation*, 2006.

[106] D. Sinkovits, "Flocking behaviour," tech. rep., University of Illinois at Urbana-Champaign, United States, 2006.

[107] D. Bourg and G. Seemann, *AI for Game Developers*. O'Reilly Media, July 2004.

[108] A. Davison, *Killer Game Programming in Java*. O'Reilly Media, May 2005.

[109] H. Su, NewAuthor2, and G. Chen, "A connectivity-preserving flocking algorithm for multi-agent systems based only on position measurements," *International Journal of Computers, Communications and Control*, vol. 82, pp. 1334–1343, July 2009.

[110] H. Su, Y. Zhou, and X. Wang, "Simulation platform for flocking in multi-agent systems with a virtual leader," *Proceedings of the IEEE Systems and Information Engineering Design Symposium*, Apr. 2008.

[111] Z. Wang and D. Gu, "Distributed leader-follower flocking control," *Asian Journal of Control*, vol. 11, pp. 396–406, July 2009.

[112] C. Reynolds, "Flocks, herds, and schools: A distributed behaviour model," *Symbolics Graphics Division - Siggraph*, 1987.

[113] L. Qi-Shao and Y. Ji-Chen, "Flocking of multi-agent systems following virtual leader with time-varying velocity," *Chinese Phys. Letter*, vol. 26, no. 2, 2009.

[114] Z. Li, Y. Jia, J. Du, and S. Yuan, "Flocking for multi-agent systems with switching topology in a noisy environment," *American Control Conference*, June 2008.

[115] C. Reynolds, "Steering behaviors for autonomous characters," *Game Developers Conference*, 1999.

[116] C. Reynolds, "Flocks, herds, and schools: A distributed behavioral model," *Computer Graphics, ACM SIGGRAPH Conference Proceedings*, vol. 21, no. 4, pp. 25–34, 1987.

[117] T. Vicsek, A. Czirók, E. Ben-Jacob, I. Cohen, and O. Shochet, "Novel type of phase transition in a system of self-driven particles," *Physical Review Letter*, vol. 75, Aug. 1995.

[118] J. Toner, Y. Tu, and S. Ramaswamy, "Hydrodynamics and phases of flocks," *Annals of Physics*, vol. 318, pp. 170–244, July 2005.

[119] L. Xiaoli and X. Yugeng, "Flocking of multi-agent dynamic systems with guaranteed group connectivity," *Proceedings of the 27th Chinese Control Conference*, July 2008.

[120] A. Jadbabaie, J. Lin, and A. Morse, "Coordination of groups of mobile agents using nearest neighbor rules," *IEEE Transactions on Automatic Control*, vol. 48, no. 6, pp. 988–1001, 2003.

[121] H. Shi, L. Wang, and T. Chu, "Virtual leader approach to coordinated control of multiple mobile agents with asymmetric interactions," *Physica D,*, vol. 213, pp. 51–65, 2006.

[122] N. Leonard and E. Friorelli, "Virtual leaders, artificial potentials and coordinated control of groups," *the 40th IEEE Conference on Decision and Control*, pp. 2968–2973, Dec. 2001.

[123] Z. Wang, D. Gu, and H. Hu, "Leader-follower flocking experiments using estimated flocking center," *Proceedings of the IEEE International conference on Mechatronics and Automation*, Aug. 2009.

[124] J. Zhou, W.Yu, X. Wu, M. Small, and J. Lu, "Flocking of multi-agent dynamical systems based on pseudo-leader mechanism," *Nonlinear Sciences*, May 2009.

[125] A. Widmer and P. Franaszek, "A dc-balanced partitioned-block 8b/10b transmission code," *IEEE*, vol. 27, pp. 440–451, Sep. 1983.

[126] Lattice, "8b/10b encoder/decoder," *Lattice Semiconductor*, vol. Reference Design RD1012, Nov. 2002.

[127] J. Wu and Y. Hsu, "8b/10b codec for efficient papr reduction in ofdm communication systems," *IEEE International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 1198–1201, 2005.

[128] L. Cai, P. Kritzinger, M. Olivares, and D. Gajski, "Top-down system level design methodology using specc, vcc and systemc," *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2002.

[129] S. Sumner, "A brief overview of 10 gigabit ethernet," tech. rep., EXFO, 2005.

[130] J. Booth and J. Booth, "Systemc modeling of a parallel processor broadcast interconnection system," *SoutheastCon, 2002. Proceedings IEEE*, pp. 76–81, 2002.

[131] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. D. Micheli, "Noc synthesis flow for customized domain specific multiprocessor systems-on-chip," *IEEE Transactions on Parallel Distribution System*, vol. 16, pp. 113–129, Feb. 2005.

[132] M. Conti and D. Moretti, "System level analysis of the bluetooth standard," *IEEE DATA*, vol. 3, pp. 118–123, Mar. 2005.

[133] N. Ahmed, I. Aref, F. Rodriguez, and K. Elgaid, "Wireless channel model based on soc design methodology," *Fourth International Conference on Systems and Networks Communications (ICSNC)*, September 2009.

[134] I. Aref, N. Ahmed, F. Rodriguez, and K. Elgaid, "Rtl-level modeling of an 8b/10b encoder-decoder using systemc," *The Fifth IEEE and IFIP International Conference on Wireless and Optical Communication Networks (WOCN2008)*, 2008.

[135] P. Rashinkar, P. Paterson, and L. Singh, *System-on-a-Chip Verification: Methodology and Techniques*. 2002.

[136] M. Petrov, T. Murgan, P. Zipf, and M. Glesner, "Functional modeling techniques for a wireless lan ofdm transceiver," *IEEE International Symposium on Circuits and Systems*, vol. 4, pp. 3973–3973, May 2005.

[137] HP, *HDLC Frame Protocol Users Guide*. HP invent, 1 ed., Feb. 2004.

[138] W. H. Tranter, K. S. Shanmugan, T. S. Rappaport, and K. L. Kosbar, *Principles of Communication Systems Simulation with Wireless Applications*. first ed., 2004.

[139] W. Stallings, *Data and Computer Communications*. Pearson Prentice Hall, 2007.

[140] V. Bagad and I. Dhotre, *Data Communication and Networking*. Technical Publications Pune, Jan. 2006.

[141] A. Leon-Garcia and I. Widjaja, *Communication networks: fundamental concepts and key architectures*. McGraw Hill, 2004.

[142] B. Walke, S. Mongold, and L. Berlemann, *IEEE-802 Wireless Systems*. John Wiley and Sons, 2006.

[143] S. Iyengar, N. Parameshwaran, V. Phona, N. Balakrishnan, and C. Okoye, *Fundamentals of Sensor Network Programming - Applications and Technology*. Wiley, 2010.

[144] R. Olfati-Saber, "Flocking for multi-agent dynamic systems: Algorithms and theory," *IEEE Transactions on Automatic Control*, June 2004.

[145] H. Tanner and G. Pappas, "Flocking in fixed and switching networks," *IEEE Transactions on Automatic Control*, Apr. 2005.

[146] E. Whittaker and G. Robinson, *The calculus of observations: An introduction to numerical analysis*. 4th edition ed., 1967.

[147] J. Kenney and E. Keeping, *Mathematics of statistics*. 3rd edition ed., 1964.

[148] F. Johnston, J. Boylan, M. Meadows, and E. Shale, "Some properties of a simple moving average when applied to forecasting a time series," *Journal of the Operational Research Society*, 1999.

[149] M. Rajan, M. Chandra, L. Reddy, and P. Hiremath, "Concepts of graph theory relevant to ad-hoc networks," *International Journal of Computers, Communications and Control*, vol. III, no. ISSN 1841-9836, pp. 465–469, 2008.

[150] B. Kwak, N. Song, and L. Miller, "On the scalability of ad hoc networks: a traffic analysis at the center of a network," *Wireless Communication and Networking Conference (WCNC-2004)*, Mar. 2004.

[151] J. Osmundson and T. Huynh, "Scalability of wireless ad hoc netwroks by simulation," 2004.

[152] G. Chartrand, *Introductory graph theory*. Courier Dover Publications, 1985.

[153] R. Trudeau, *Introduction to graph theory*. Courier Dover Publications, 1994.

[154] L. Ding and Z.-H. Guan, "Modeling wireless sensor networks using random graph theory," *Elsevier B.V.*, 2008.

[155] M. A. Rajan, M. G. Chandra, L. C. Raddy, and P. Hiremath, "Concepts of graph theory relevant to ad-hoc networks," *International Journal of Computers, Communications and Control*, 2008.

[156] J. Prizmic and R. Podgornik, "Models of the small world," 2001.

[157] D. J. Watts and S. H. Strogatz, "Collective dynamics of small-world networks," 1998.

[158] E. Weisstein, "Adjacency matrix," tech. rep., Math-World, 1999.

[159] B. Sklar, *Digital Communications:Fundamentals and Applications*. Prentice Hall, Jan. 2001.

[160] D. Falconer, F. Adachi, and B. Gudmundson, "Time division multiple access methods for wireless personal communications," *IEEE Communication Magazine*, Jan. 1995.

[161] J. Kurose and K. Ross, *Computer Networking: A Top-Down Approach*. Addison-Wesley, fifth ed., 2009.