



<https://theses.gla.ac.uk/>

Theses Digitisation:

<https://www.gla.ac.uk/myglasgow/research/enlighten/theses/digitisation/>

This is a digitised version of the original print thesis.

Copyright and moral rights for this work are retained by the author

A copy can be downloaded for personal non-commercial research or study, without prior permission or charge

This work cannot be reproduced or quoted extensively from without first obtaining permission in writing from the author

The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the author

When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given

Enlighten: Theses

<https://theses.gla.ac.uk/>
research-enlighten@glasgow.ac.uk

Modelling and Analysis of Next Generation Home Networks

Duncan J McLaren

A Thesis Submitted To
The Universities of

Edinburgh
Glasgow
Heriot-Watt
Strathclyde

For the Degree of
Doctor of Engineering in System Level Integration

© D. J. McLaren
February 2007

ProQuest Number: 10800612

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



ProQuest 10800612

Published by ProQuest LLC (2018). Copyright of the Dissertation is held by the Author.

All rights reserved.

This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

GLASGOW
UNIVERSITY
LIBRARY:

Abstract

As Home Networking grows over the next 20 years the need for accurate models for both the network and the hardware becomes apparent. In this work, these two areas are considered together to develop a combined hardware and network model for a HomePlug power line based network. This change of focus is important when the type of devices that will be running on tomorrow's home network is considered. It will have evolved from a simple network of PCs sharing an Internet connection to a large heterogeneous structure of embedded System-on-Chip devices communicating on a variety of linked network technologies.

This work presents a novel combined hardware and network modelling tool that address the following areas:

1. Development of a system level model of a HomePlug power-line based network, including the fundamental network protocols, the SoC hardware and the physical channel.
2. Use the developed model to explore various system scenarios.
3. Development of alternative hardware algorithms within the design.

The model developed uses a Discrete Event simulation method to allow designers to explore areas such as:

1. How does the networking hardware (i.e. the components on the SoC) interact, and what are the issues of changing the algorithms.
2. How do the nodes on the network interact, as the traffic patterns are different to those found on traditional (office-based) networks, as there will be a greater amount of streaming media.

Acknowledgements

I would like to thank everyone who has helped me in completing this work. In no particular order:

Jim Herd, Jim Mellon and James McClean for all their academic, technical and moral support throughout the last 5 years. Their advice and support has been invaluable, and has really helped me to complete the work.

Sandie Buchanan and all the staff at the ISLI, without whom I wouldn't have started the course. Tony Kirkham for letting me moan about things, and offering advice when I asked for it.

And finally to my friends and family. I owe a huge debt of gratitude to Gareth. His support and a friendly shoulder to cry on at various times have been very much needed and appreciated, along with the friendly kick up the backside when needed! I also need to thank my parents, Helen and John, and my sister Catriona who have put up with me as I've completed this. I haven't been the nicest person to live with and their patience and understanding has been a huge support to me through all the highs and lows. This work is as much for them as it is for me.

And finally to the EPSRC and Cadence who funded me through the research, I would like to say thank you.

Author's Declaration

No portion of work contained in this thesis has been submitted in support of any application for any other degree or qualification of this or any other university or institute of learning.

I declare that the work presented in this thesis is entirely my own contribution, unless otherwise stated.

D J McLaren

October 2005

Table of Contents

Abstract.....	i
Acknowledgements.....	ii
Author's Declaration.....	iii
Table of Contents.....	iv
Table of Figures.....	vii
Table of Tables.....	xi
Abbreviations Used.....	xiv
Chapter 1 – Introduction.....	1
1.1 Introduction.....	1
1.2 Overview.....	3
1.3 Aims.....	10
1.4 Summary.....	11
Chapter 2 - Literature Review.....	12
2.1 Introduction.....	12
2.2 Network Modelling.....	13
2.2.1 Overview.....	13
2.2.2 Types of Network Model.....	14
2.2.3 Languages Used to Model Networks.....	15
2.2.4 Power Line Network Modelling.....	15
2.2.5 Network Modelling Summary.....	17
2.3 Hardware Modelling.....	18
2.3.1 Overview.....	18
2.3.2 Languages.....	19
2.3.3 Simulation Issues.....	22
2.3.4 Summary of Hardware Modelling/Simulation.....	23
2.4 Summary.....	25
Chapter 3 - Background Theory.....	26
3.1 Introduction.....	26

3.2	Computer Networking.....	27
3.2.1	History.....	27
3.2.2	OSI Model.....	27
3.2.3	IEEE Networking Standards.....	29
3.3	HomePlug.....	31
3.3.1	Overview.....	31
3.3.2	HomePlug MAC.....	33
3.3.3	HomePlug PHY.....	45
3.4	Channel Model.....	53
3.4.1	Transfer Function.....	53
3.4.2	Noise Model.....	56
3.5	Summary.....	59
Chapter 4 - Modelling Environment Requirements and Control.....		60
4.1	Introduction.....	60
4.2	Requirements.....	61
4.3	Top Level Model Structure.....	64
4.4	Message/Event System.....	67
4.4.1	Event System Overview.....	67
4.4.2	System Events and Sequences.....	69
4.4.3	Event Structure.....	74
4.4.4	Event Handling Within Node-Threads.....	77
4.5	System Controller.....	81
4.5.1	Overview.....	81
4.5.2	Command File Parsing.....	81
4.5.3	System Controller Event System.....	83
4.5.4	System Controller Event Processing Algorithms.....	85
4.6	Node-Thread Event Handling.....	88
4.6.1	SoC Node-Thread.....	88
4.6.2	MAC Node-Thread.....	90
4.6.3	PHY Node-Thread.....	93
4.7	Summary.....	96
Chapter 5 - Modelling Environment HomePlug Components.....		97

5.1	Introduction.....	97
5.2	MAC Model.....	98
5.2.1	MAC Frame Assembler (assemble).....	99
5.2.2	MAC Frame Re-Assembler (disassemble).....	104
5.3	PHY Model.....	108
5.3.1	PHY Encoder.....	109
5.3.2	Frame Control FEC Encoder (fc_encode).....	111
5.3.3	Payload FEC Encoder (data_encode).....	115
5.3.4	OFDM Encoder (ofdm_encode).....	124
5.3.5	PHY Decoder.....	130
5.3.6	OFDM Decoder (ofdm_decode).....	134
5.3.7	Frame Control Decoder (fc_dec).....	140
5.3.8	Payload Decoder (payload_dec).....	144
5.4	Channel Model.....	155
5.5	Summary.....	158
Chapter 6 - Results.....		159
6.1	Introduction.....	159
6.2	Software Suite.....	160
6.3	Typical Use Case.....	161
6.4	Throughput Verses Number of Nodes.....	163
6.5	Latency Verses Buffer Size.....	165
6.6	Summary.....	167
Chapter 7 - Conclusions.....		168
7.1	Introduction.....	168
7.2	The Problem.....	169
7.3	The Solution.....	171
7.4	Evaluation of Work.....	174
7.5	Future Work.....	176
7.6	Summary.....	178
Chapter 8 - References.....		179

Table of Figures

Figure 1.1 – Broadband Internet Penetration in major economies	3
Figure 1.2 – Home Network Market Value.....	4
Figure 1.3 – Tomorrows Home Network.....	7
Figure 1.4 – System-on-Chip Block Diagram.....	8
Figure 2.1 – Layers of Abstraction in Hardware Modelling	19
Figure 2.2 – C++ Based Hardware Design Flow.....	20
Figure 3.1 – Basic Model Block Diagram	26
Figure 3.2 – Two Devices Communicating Using the OSI 7-Layer Model	28
Figure 3.3 – Data Encapsulation	29
Figure 3.4 – Network Protocol Stack.....	29
Figure 3.5 – ETSI & HomePlug Power-line Frequency Ranges	32
Figure 3.6 – HomePlug Logical Network	33
Figure 3.7 – Frame Assembly Sequence	35
Figure 3.8 – Service Block Structure	36
Figure 3.9 – MAC Frame Structure	37
Figure 3.10 – Frame Control Structure	38
Figure 3.11 – Segmentation Process	39
Figure 3.12 - Response Frame Control	39
Figure 3.13 - Response Frame Timing	41
Figure 3.14 - Channel Access Process	42
Figure 3.15 - Channel Access Timing	44
Figure 3.16 - PHY Transmitter Block Diagram	45
Figure 3.17 - Frame Control FEC Block Diagram	47
Figure 3.18 - Product Encoder Matrix	47
Figure 3.19 - Frame Control Interleaver Bit Spreading.....	48
Figure 3.20 - Payload FEC Block Diagram	48
Figure 3.21 - Scrambler Block Diagram	49
Figure 3.22 - Reed-Solomon Encoder Block Diagram	49
Figure 3.23 - Convolutional Encoder Block Diagram	50

Figure 3.24 - Bit Puncturer	50
Figure 3.25 - OFDM Encoder Block Diagram	51
Figure 3.26 – Channel Model	53
Figure 3.27 - Two-port Model.....	53
Figure 3.28 - Simple Network Model	55
Figure 3.29 – Noise Model	56
Figure 3.30 – Extended Noise Model	57
Figure 4.1 – Model Structure	64
Figure 4.2 – Event Communications	68
Figure 4.3 – Phase 1 Event Sequence	71
Figure 4.4 – Phase 2 Event Sequence	71
Figure 4.5 – Phase 3 Event Sequence	72
Figure 4.6 – Phase 4 Event Sequence	72
Figure 4.7 – Phase 5 Event Sequence	73
Figure 4.8 – Example Command File	81
Figure 4.9 – System Controller Operation.....	84
Figure 4.10 – Outgoing Event Nassi-Shneiderman Diagram 1	85
Figure 4.11 – Outgoing Event Nassi-Shneiderman Diagram 2	85
Figure 4.12 – Outgoing Event Nassi-Shneiderman Diagram 3	86
Figure 4.13 – Incoming Event Nassi-Shneiderman Diagram 1	86
Figure 4.14 – Incoming Event Nassi-Shneiderman Diagram 2	87
Figure 4.15 – SoC Event Nassi-Shneiderman Diagram 1	89
Figure 4.16 – SoC Event Nassi-Shneiderman Diagram 2	89
Figure 4.17 – MAC Event Nassi-Shneiderman Diagram 1	91
Figure 4.18 – MAC Event Nassi-Shneiderman Diagram 2	91
Figure 4.19 – MAC Event Nassi-Shneiderman Diagram 3	92
Figure 4.20 – PHY Event Nassi-Shneiderman Diagram 1	94
Figure 4.21 – PHY Event Nassi-Shneiderman Diagram 2	94
Figure 4.22 – PHY Event Nassi-Shneiderman Diagram 3	94
Figure 4.23 – PHY TRANSMIT Event Nassi-Shneiderman Diagram	95
Figure 4.24 – PHY NEW_DATA Event Nassi-Shneiderman Diagram	95
Figure 5.1 – MAC Structure Chart	98

Figure 5.2 – Service Block	100
Figure 5.3 – PHY Frame Format	102
Figure 5.4 – Service Block Re-creation Process	106
Figure 5.5 – PHY Structure Chart	108
Figure 5.6 – PHY Encoder Structure Chart	109
Figure 5.7 – PHY Encoder Data Flow	109
Figure 5.8 – Frame Control FEC Block diagram	111
Figure 5.9 – Product Encoder Matrix	112
Figure 5.10 – Product Encoder Parity Generation	113
Figure 5.11 – Symbol Generator Process	114
Figure 5.12 – Payload FEC Encoder Block Diagram	115
Figure 5.13 – Scrambler Circuit	118
Figure 5.14 – Reed Solomon Encoder Circuit	119
Figure 5.15 – Convolutional Encoder Circuit	120
Figure 5.16 – Bit Puncturing	121
Figure 5.17 – Interleaver Process	122
Figure 5.18 – OFDM Encoder	124
Figure 5.19 – Modulation Constellations	126
Figure 5.20 – Mapper Operation	126
Figure 5.21 – IFFT Operation	127
Figure 5.22 – Cyclic Prefix	128
Figure 5.23 – Pulse Shaper	129
Figure 5.24 – PHY Decoder Structure Chart	130
Figure 5.25 – PHY Decoder Data Flow	130
Figure 5.26 – OFDM Decoder	134
Figure 5.27 – Cyclic Prefix Remover	135
Figure 5.28 – Channel Filter Circuit	137
Figure 5.29 – De-Mapping Operation	138
Figure 5.30 – Frame Control FEC Decoder Block Diagram	140
Figure 5.31 – Product Decoder Operation	142
Figure 5.32 – Payload FEC Decoder Block Diagram	144
Figure 5.33 – De-Puncture Operation	148
Figure 5.34 – Reed Solomon Decoder Block Diagram	151
Figure 5.35 – Channel Model	156

Figure 5.36 – Channel Model Message Passing	156
Figure 6.1 – Typical Home Network Scenario	161
Figure 6.2 – Home Network Scenario Simulation Output	162
Figure 6.3 – Throughput Simulation Setup	162
Figure 6.4 – Throughput Simulation Graph	164
Figure 6.5 – Latency Simulation Setup	165
Figure 7.1 – Model Structure	171
Figure 7.2 – SoC Design Flow	174

Table of Tables

Table 1.1 – Chapter Contents	2
Table 1.2 – Power-line Networking Standards	5
Table 1.3 – 802.11 Technologies	6
Table 3.1 – Description of OSI Layers	28
Table 3.2 – IEEE 802 Sub-Groups.....	30
Table 3.3 – Segment Control Field Structure	37
Table 3.4 - MAC Management Entries	45
Table 3.5 - Blocked HomePlug Frequencies	46
Table 3.6 - Bits Per Carrier for HomePlug Modulations	46
Table 3.7 - Reed-Solomon Modes.....	49
Table 3.8 - Multi-paths Through Network Model	55
Table 3.9 – Noise Characteristics	57
Table 4.1 – Modelling System Mandatory Requirements	62
Table 4.2 – Modelling System Non-Mandatory Requirements	63
Table 4.3 – Model Terminology	64
Table 4.4 – Model Command File Instructions	65
Table 4.5 – Modelling System Events	70
Table 4.6 – Event Structure	74
Table 4.7 – Event Data Structure	75
Table 4.8 – Event Linked List Functions	78
Table 4.9 – SoC Thread Package Structure	88
Table 4.10 – SoC Events	88
Table 4.11 – MAC Thread Package Structure	90
Table 4.12 – MAC Events	90
Table 4.13 – PHY Thread Package Structure	93
Table 4.14 – PHY Events	93
Table 5.1 – MAC Frame Assembler Inputs and Outputs	99
Table 5.2 – MAC Service Block Inputs and Outputs	100

Table 5.3 – Segmentation Information Inputs and Outputs	101
Table 5.4 – MAC Service Block Inputs and Outputs	102
Table 5.5 – MAC Frame Re-Assembler Inputs and Outputs	104
Table 5.6 – MAC Service Block Inputs and Outputs	105
Table 5.7 – MAC Service Block Inputs and Outputs	106
Table 5.8 – MAC Service Block Recreation Inputs and Outputs	107
Table 5.9 – PHY Encoder Inputs and Outputs.....	109
Table 5.10 – Frame Control FEC Input and Outputs	111
Table 5.11 – Product Encoder Input and Outputs	112
Table 5.12 – Frame Control Interleaver Input and Outputs.....	113
Table 5.13 – Payload FEC Inputs and Outputs	115
Table 5.14 – Scrambler Function Inputs and Outputs	118
Table 5.15 –Reed-Solomon Encoder Function Inputs and Outputs	119
Table 5.16 – Convolutional Encoder Inputs and Outputs	120
Table 5.17 – Bit Puncture Inputs and Outputs	121
Table 5.18 – Interleaver Inputs and Outputs	122
Table 5.19 – OFDM Encoder Inputs and Outputs	124
Table 5.20 – Mapper Inputs and Outputs	126
Table 5.21 – IFFT Inputs and Outputs	128
Table 5.22 – Cyclic Prefix Inputs and Outputs	128
Table 5.23 – Cyclic Prefix Inputs and Outputs	129
Table 5.24 – Frame Control Decoder Inputs and Outputs	131
Table 5.25 – Payload Decoder Inputs and Outputs	131
Table 5.26 – Channel Estimator Inputs and Outputs	132
Table 5.27 – OFDM Decoder Inputs and Outputs	134
Table 5.28 – Cyclic Prefix Remover Inputs and Outputs	136
Table 5.29 – FFT Block Inputs and Outputs	136
Table 5.30 – Channel Filter Inputs and Outputs	137
Table 5.31 – De-Mapper Inputs and Outputs.....	138
Table 5.32 – Frame Control Decoder Inputs and Outputs	140
Table 5.33 – Bit Generator Inputs and Outputs	141
Table 5.34 – De-Interleaver Inputs and Outputs	142
Table 5.35 – Product Decoder Inputs and Outputs	142
Table 5.36 – Payload Decoder Inputs and Outputs	144

Table 5.37 – Payload De-Interleaver Inputs and Outputs	147
Table 5.38 – De-Puncturer Inputs and Outputs	149
Table 5.39 – Viterbi Decoder Inputs and Outputs	149
Table 5.40 – Reed-Solomon Decoder Inputs and Outputs	151
Table 5.41 – Scrambler Function Inputs and Outputs	153
Table 6.1 – Channel Characteristics.....	160
Table 6.2 – Network Traffic.....	161
Table 6.3 – Home Network Scenario Results.....	162
Table 6.4 – Throughput Simulation Results.....	163
Table 6.5 – Latency Simulation Traffic.....	165
Table 6.6 – Latency Simulation Results.....	166
Table 7.1 – Modelling System Mandatory Requirements	173

Abbreviations Used

ACK	Acknowledgement
AHB	Advanced High-performance Bus
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
ARM	Advanced RISC Machines
ARQ	Automatic Repeat Request
ASIC	Application Specific Integrated Circuit
AV	Audio-Visual
B-PAD	Block Pad
BPSK	Binary Phase Shift Keying
CAP	Channel Access Priority
CIFS	Contention Inter-frame Space
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Processing Unit
CRC	Cyclic-Redundancy Check
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
DBPSK	Differential Binary Phase Shift Keying
DCF	Distributed Control Function
DES	Data Encryption Standard
DQPSK	Differential Quadrature Phase Shift Keying
DSP	Digital Signal Processing
EDA	Electronic Design Automation
E-PAD	Encryption Pad
ETSI	European Telecommunications Standards Insitutue
FCC	Federal Communications Commission
FCS	Frame Check Sequence
FEC	Forward Error Correction
FFCS	Frame Control Frame Check Sequence
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FTP	File Transfer Protocol
GCC	Gnu C Compiler
GF	Galois Field
GFSK	Gaussian Frequency Shift Keying
GPIO	General Purpose Input/Output
HAVi	Home Audio Video Interoperability
HDL	Hardware Description Language
HPNA	Home Phone-line Network Alliance
HTTP	Hyper-Text Transfer Protocol
I2C	Intelligent Interface Controller
ICV	Integrity Check Value
IEEE	Institute of Electrical and Electronic Engineers
IFFT	Inverse Fast Fourier Transform
IP	Intellectual Property/Internet Protocol
IRDA	Infrared Data Association
ISM	Industrial, Scientific and Medical
ISO	International Standards Organisation
LAN	Local Area Network
LLC	Logical Link Control

MAC	Media Access Controller
MAN	Medium Area Network
Mb/s	Mega-bits per Second
MHz	Mega-hertz
MOC	Methods of Computation
MPDU	MAC Protocol Data Unit
MPEG	Moving Picture Experts Group
MSDU	MAC Service Data Unit
NACK	Negative Acknowledgement
OFDM	Orthogonal Frequency Division Multiplexing
OOK	On-Off Keying
OSI	Open Systems Interconnection
PAN	Personal Area Network
PCS	Physical Carrier Sense
PSD	Power Spectral Density
PHY	Physical Layer
PRS	Priority Resolution Symbol
QoS	Quality of Service
RAM	Random Access Memory
RIFS	Reduced Inter-Frame Space
ROBO	Robust OFDM
ROM	Read Only Memory
RS	Reed-Solomon
RTL	Register Transfer Level
SMB	Simple Message Block
SMTP	Simple Mail Transfer Protocol
SoC	System on Chip
SPICE	Simulation Program With Integrated Circuit Emphasis
SSH	Secure Socket Shell
TAP	Test Access Point
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UPnP	Universal Plug-n-Play
VCS	Virtual Carrier Sense
VHDL	Very High Speed Hardware Description Language
VLAN	Virtual LAN
WAN	Wide Area Network
WLAN	Wireless LAN
XML	Extendible Mark-up Language

Chapter 1 - Introduction

Introduces the thesis, giving the reader the necessary background to the problem that is being addressed, as well as the content of the rest of the Thesis. It also gives the main aims of the research.

1.1 INTRODUCTION

Many of the concepts and ideas that are the basis of the work carried out will be introduced, and the context set for the work. This is necessary due to the large size of the areas being considered within the work, namely home networking, network modelling and hardware modelling. Traditionally these have been considered separately; however as the usage of networking within the home increases, the need to consider these together becomes apparent. This is a concept which is expanded upon in subsequent chapters.

A brief overview of home networking is given, highlighting some of the business issues which are important to consider alongside the technical ones. It also gives a summary of the various technologies used within the home. It introduces some of the challenges resulting from the likely hardware that the network technology will be implemented upon, which is likely to be System-on-Chip (SoC), that by its very nature is resource constrained and presents unique challenges of its own.

The structure of the thesis is as follows:

Chapter 1	Introduces the thesis, giving the reader the necessary background to the problem that is being addressed, as well as the content of the rest of the Thesis. It also gives the main aims of the research
Chapter 2	A walk-through of the relevant literature on the topics of network and hardware modelling, giving some of the important concepts, and how they can be mapped onto the problem situation being looked at here.

Chapter 3	Provides a more detailed description of the theory and standards that were used in creating the system model. It starts with a brief history of computer networking and highlights some of the relevant standards. The HomePlug power-line standard is then described in some detail, before a brief overview of the channel model is given
Chapter 4	Describes the model that was developed. It first gives the requirements of the model (based on the discussions of previous chapters), and then describes the structure of the model, and how it is controlled. It also introduces the terminology used to describe aspects of the model.
Chapter 5	Continues the description of the model, with the focus being on the way the data is modified by the algorithms. It starts with a description of the MAC functions, before describing the PHY and finally the channel.
Chapter 6	Provides the results of using the model developed to explore some simple use cases. It also shows how the model can be used to explore alternative algorithms.
Chapter 7	Concludes the work presented previously, by giving a summary of the problem area and then the model that was developed. It finishes with a discussion of the future work that could be carried out.

Table 1.1 – Chapter Contents

1.2 OVERVIEW

Home networking, as we know it, is likely to change dramatically over the next 15-20 years. Today, when we talk about home networking we generally mean two (or more) computers connected together via Ethernet or Wi-Fi to share an Internet connection and peripherals such as printers. This need to share an Internet connection has grown out of the increased adoption of broadband throughout the western world, as shown in Figure 1.1 [1].

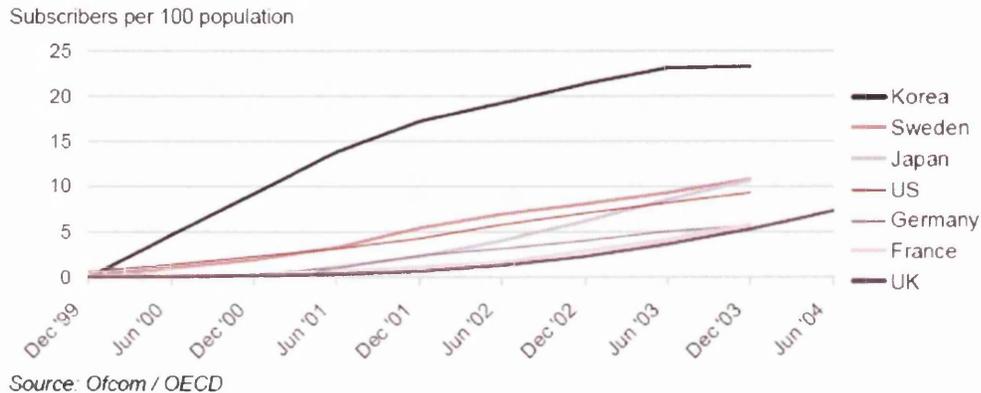


Figure 1.1 – Broadband Internet Penetration in major economies

This trend is likely to continue, yet future growth in home networking will not only be driven by this need but also by the need to share digital multimedia traffic throughout the home. Although many people are predicting that tomorrow's home network will be used mainly for streaming media throughout the home [2, 3], it is equally probable that it will integrate many currently un-connected devices such as home entertainment, white-goods, PCs (in whatever form they take), home automation, communications, etc. This integration of many disparate devices will lead to many different types of traffic on the network(s) from high bandwidth, low latency video traffic to low bandwidth higher latency automation traffic. Many in industry are also suggesting the demise of the PC as it is known today. In its place will be a "Home Gateway" [4] which will act as a server in many respects, by piping the incoming broadband stream throughout the home and ensuring that the internal traffic is routed to the appropriate location within the home. If someone wants access to the Internet say, they will use a terminal device (for example a PDA or display screen) and the information will be routed via the gateway to and from the Internet. One important factor of the gateway is it must be easy to maintain and upgrade, either by the homeowner or by a service provider.

Another differentiating aspect of future home networks will be the actual network technology, as many homeowners will be unwilling or unable to run Cat5 Ethernet cabling throughout their home. To overcome this problem, the electronics industry has been proposing so-called “no-new-wires” technologies which make use of either the home’s existing wiring (i.e. the phone- and power-lines) or wireless. Figure 1.2 shows the predicted market value of the home networking sector up to 2002, broken down by technology [5]. Although the figures suggest that phone line will be the most predominant, time has shown that this is not the case and that wireless has become the predominant home networking technology. Unfortunately further data on the market value was not obtained.

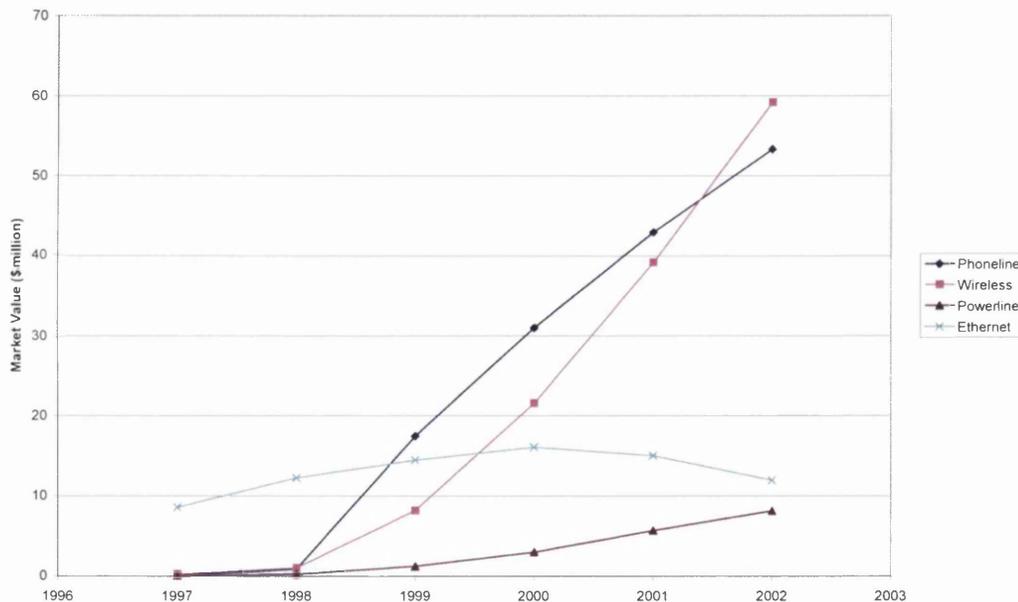


Figure 1.2 – Home Network Market Value

Phone-line networking uses the existing phone lines within the house to transport data, and the main standard is HomePNA (Home Phone-line Networking Alliance) [6]. Version 1 of the standard was developed from technology produced by Tut Systems in the mid-90’s, and was capable of 1Mb/s using the 4-10MHz frequency range (much higher than the analogue voice signals). Version 1 has an effective range of 150m, and it will support 25 connections. Version 2 was released around 2000, from technology developed by Broadcom, and it is capable of 10Mb/s with a range of 350m.

The primary power-line networking standard is HomePlug. HomePlug is an industrial consortium made up of many key players who released V1 in July 2001 [7]. V1 allows speeds up to 14Mb/s, although true speed is somewhere between 4 and 5 depending on channel conditions. The technology is based around OFDM and the standard defines both the Physical and Media-access layers (further details are provided in Chapter 3). They are currently working on HomePlug AV, which is a 100-200 Mb/s system aimed at audio/visual networks. Again this is based on OFDM, using more sub-carriers and larger bandwidth than the current system, along with a re-designed MAC layer. There have been many different power-line standards developed over the years. In the past these have mainly focused on home automation, rather than home networking, however recent developments in ASIC and DSP have enabled more advanced modulation techniques that allow true networking over the power-line. Table 1.1 summarises some of the main technologies developed [8].

Technology	Primary Use	Data Rates	Frequency Range	Modulation
HomePlug	Data Comms	14 Mb/s	4.47 to 20.1 MHz	OFDM with BPSK, DBPSK, DQPSK or ROBO
X-10	Home Automation	50 or 60 b/s	120 kHz	
CEBus	Home Automation	8.5 kb/s	100 to 400 kHz	Spread Spectrum
LONWorks	Industrial Control	10 kb/s, 5 kb/s or 2 kb/s	100 to 450 kHz, 125 to 140 kHz, or 9 to 95 kHz	Spread Spectrum
Mainnet	Data Comms	2.5 Mb/s		
nSine	Data Comms	2.5 Mb/s (40 Mb/s planned)	8 to 32 MHz	WAM with OOK or GFSK
ds2	Data Comms	45 Mb/s	1 to 38 MHz	OFDM
Ascom	Data Comms	3 Mb/s		

Table 1.2 – Power-line Networking Standards

Wireless networking is primarily provided by two different standards, IEEE 802.11 and Bluetooth. Bluetooth is a lower rate system, with a typical device operating at speeds of up to 1Mb/s over a range of 10m. It is commonly used to create Personal Area Networks (PANs), where small mobile devices are connected together such as a mobile telephone and a hands-free headset. The channel can support both synchronous (voice) and asynchronous (data) traffic. The 802.11 standards are used to create Wireless LANs

and have a much higher data rate and range. There are three different versions currently, 802.11a, 802.11b and 802.11g and Table 1.2 summarises the main features of each.

Technology	Data Rates	Range (Inside/Outside)	Frequency Range	Modulation
802.11a	Up to 54 Mb/s	16m / 33m	5.4GHz ISM Band	Coded OFDM
802.11b	Up to 11 Mb/s	50m / 100m	2.4GHz ISM Band	Spread Spectrum
802.11g	Up to 54 Mb/s	50m / 100m	2.4GHz ISM Band	Spread Spectrum (up to 11 Mb/s) OFDM (11 to 54 Mb/s)

Table 1.3 – 802.11 Technologies

At present 802.11g is becoming the dominant standard, as it is faster than 802.11b (which was the primary standard, as the cards were cheaper than the faster 802.11a devices). 802.11g uses the same frequency range as 802.11b, but has the speed of 802.11a, which is allowing companies to roll out the faster 11g networks whilst still maintaining backwards compatibility with the 11b devices they already have. Work is currently underway to develop the next generation of wireless networks (802.11n) and this is due to be released sometime in the next few years.

While each of these technologies can solve the problem of connecting devices, they cannot satisfy the needs of tomorrow's home network. Although each has strengths, they also each have weaknesses, and it is therefore likely that a hybrid system will be adopted that uses the technology where it is most appropriate. For example, a wall-mounted plasma screen doesn't really need to be connected via wireless, and so would probably use the power-line. A potential home networking scenario is shown in Figure 1.3.

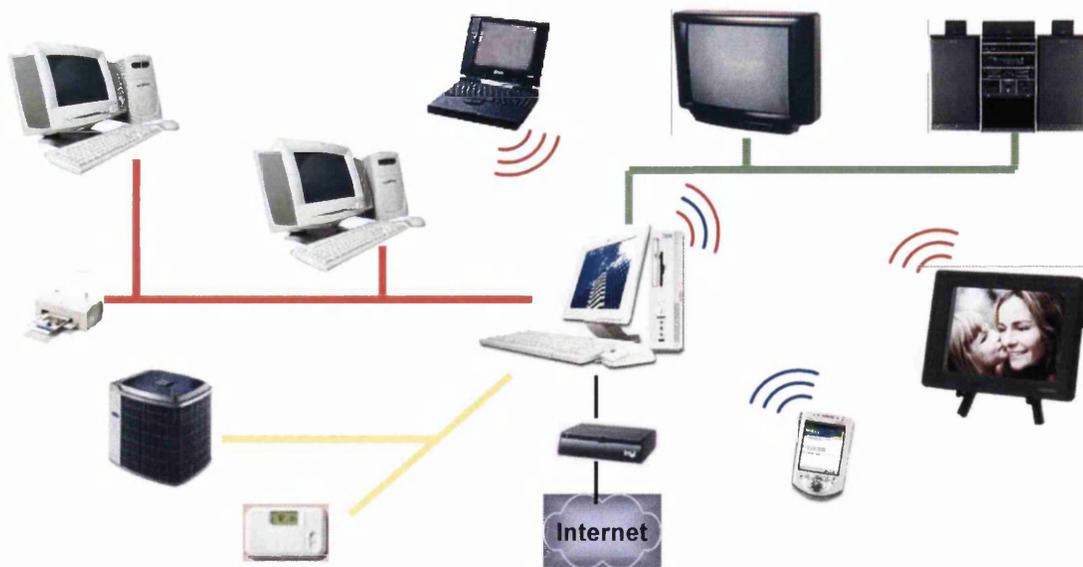


Figure 1.3 – Tomorrows Home Network

As mentioned above, future home networks are likely to integrate many different devices. The technology that will allow this is System-on-Chip (SoC), as this will enable the networking hardware to be integrated with the rest of the system. For example, a networked display screen (i.e. a large plasma) would have an integrated SoC which would be responsible for displaying the correct information on the display screen (probably using a hardware MPEG decoder for digital video), communicating on the network (whichever one is being used), interacting with the user, etc. An example System-on-Chip is shown in Figure 1.4.

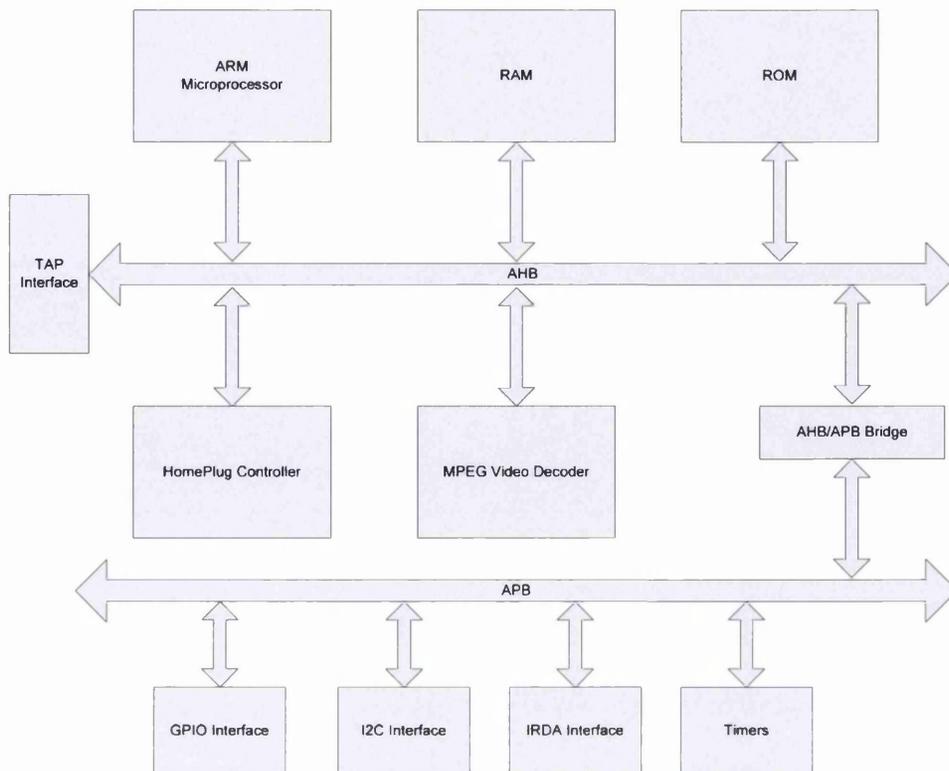


Figure 1.4 – System-on-Chip Block Diagram

This adds two further potential complications to the network. The first is how to get these different devices to communicate with each other. The second being how the use of the SoC will impact on the operation/performance of the system, as a SoC is more resource limited than a general purpose computer.

There has been much academic and industrial research into how to get disparate devices to communicate with each other. Currently the main contenders are UPnP [9] (Universal Plug'n'Play) and HAVi [10] (Home Audio Video interoperability). These efforts have mainly focused at the higher layers of the network protocols, as it is the higher layers where the data starts to take meaning.

Not as much focus has been made on how the lower levels of this type of networking system interact as this has traditionally been seen as two separate areas, one being the hardware itself, the other being the network. In this type of network, the issue is more how both the hardware and the network interact.

A standard design practice is to develop a software model of the hardware components. This gives the design engineer the ability to explore various alternative hardware algorithms very quickly, as well as providing a golden reference model for the hardware verification stage of the design. In the past the model has been either a hardware model or a network model, but without consideration of the interaction between the two parts (i.e. the network model will not model how the actual hardware running the protocol will operate). By separating the hardware from the network, system designers are not able to explore how the whole system interacts. In the complex network described, this could pose a major shortfall, as the designers will be unable to evaluate the effects different hardware algorithms might have on the operation of the system as a whole.

The final part of the problem is the communication channels themselves. Using the “no-new-wires” technologies, these are going to be some of the noisiest and most hostile transmission channels available. In the past channel models have tended to be either statistical models (i.e. so many packets will be in error over a particular period of time) or focused on frequency ranges outwith those that are used for the new standards (i.e. a power-line model that is used to determine the loss characteristics of the mains signal). In order for the system level model to be of any use, accurate, or at least adequate, channel models are needed.

1.3 AIMS

Given the issues and problems raised in the previous section, the aims of this research are:

- To develop a system level model for the next-generation home network, as described above. This will focus on one particular technology, the HomePlug V1.0 Power-line standard, to provide a proof of concept. The model will:
 - Model the Soc, MAC and PHY components of each node in the network
 - Model the communications channel
 - Follow the modulation/encoding mechanism for HomePlug
 - Follow the HomePlug channel access mechanism
- Using the model developed, run various scenarios to explore how the system will operate under normal conditions
- Use the model to explore abnormal conditions, i.e. under what conditions will the network fail
- Develop alternative hardware algorithms for components within the design

1.4 SUMMARY

A brief overview of home networking has been given, focusing on the technologies involved and the challenges faced in developing products in what is a rapidly growing market. As the market has grown there is an increasing need to develop products for it. However as it is a consumer electronics market, these products need to be easy to use for the end user. This need is one of the drivers for developing models of these systems, so that developers have a better understanding of the issues involved.

The concepts introduced here are expanded upon in subsequent chapters to show why a model is relevant, and what the requirements of any such model would be. These requirements are then used to develop the model which is the focus of the work presented here.

Chapter 2 - Literature Review

A walk-through of the relevant literature on the topics of network and hardware modelling, giving some of the important concepts, and how they can be mapped onto the problem situation being looked.

2.1 INTRODUCTION

The work presented covers two areas, namely network modelling and hardware modelling/simulation. This in itself poses a problem as the two areas have traditionally been treated separately, however as the number of networked devices increases, and the fact that a lot of these will be smaller embedded SoC-type devices, the need to consider network and hardware modelling/simulation together becomes apparent.

Network modelling will be explored in the first section and some of the techniques and methods are explained. This section looks at the choice of what to model and how to model it, for example what language to use. In the next section, the focus is on hardware modelling and simulation and it introduces some of the key points. In the final section the findings of the network and hardware modelling sections are compared and a combined approach to network and hardware modelling is proposed.

2.2 NETWORK MODELLING

2.2.1 OVERVIEW

There are three main ways to model a network.

1. Analytical Based.
2. Simulation Based.
3. Emulation Based.

An analytical model uses mathematical equations or Markov Models to describe how the data is affected as it progresses through the model. Often these are compared with experimental results done on real systems to provide some level of validity to the model. They tend to be more suited to modelling how the data is modified within the system.

A simulation based model uses discrete events to determine when things happen within the model. At these times, data can be updated, using either an analytical based equation or an algorithm that describes what happens to the data. These types of models are better suited to modelling data transactions and interactions between “devices”.

An emulation model uses real hardware to model different network protocols using existing (known) protocols. For example, in [11] an extension to the Linux kernel is used to emulate various protocols using an Ethernet LAN. These types of model do however require much more hardware than an analytical or simulation based model.

Early attempts at modelling networks tended to focus on analytical approaches, for example [12] uses an analytical approach to model a packet radio system, and [13] uses it to model Radio-Frequency, Laser and Satellite based networks. The reason for the early use of analytical models is the computational intensive nature of simulation [12].

Analytical approaches are still widely used as they are useful when exploring a particular aspect of a network, for example [14] uses an analytical method to explore the Distributed Control Function (DCF) of the 802.11 Media Access Control (MAC). When looking at more than one aspect of a network however, simulation is better suited as it is easier to integrate the effects of concurrent events. In [15], [16] and [17] for example, the authors use both simulation based modelling and analytical based modelling.

Simulation models are more widely used when modelling either many layers within the network protocol stack, as in [18] and [19] or when modelling networks where multiple nodes are involved, as in [20] and [21].

As well as there being different ways to model a network, there are different aspects within the network to model. These can range from the channel itself, as in [22] and [23] through the lower layers of the network protocol (as in [24]) to modelling higher level issues such as routing and traffic (as in [25]). The focus of the model often impacts the choice of modelling technique; however, often the same area can be modelled in different ways. For example, the Distribution Control Function (DCF) of the 802.11 MAC was modelled using an analytical approach in [14], but was also modelled using a simulation based approach in [21].

There is also the question of how many nodes are in the networks being modelled as this can have an impact on the type of model used. The common theme seems to be to use an analytical approach for simple point-to-point type networks, but simulation for multi-point networks.

2.2.2 TYPES OF NETWORK MODEL

This section attempts to give an overview of the range of network types that have been modelled. The purpose is to show the wide range that has been covered over the years. Even using the papers referenced in this section ([11] through [41]) as a small sample, a wide variety of networks have been modelled, including (but not limited to)

- Wireless LAN (802.11x, Bluetooth, etc.)
- ATM (Telephony backbone protocol)
- LANs (Ethernet, Token Ring, etc.)
- Power and Phone Line
- Wide Area Networks

The focus of much of the research has been in wireless technologies. This is probably due to the ever increasing popularity of these networks, and the amount of research

(both Academic and Industrial) that is occurring. There is a lot of benefit from getting a better understanding of this environment as it allows better data rate, improved Quality of Service, etc.

2.2.3 LANGUAGES USED TO MODEL NETWORKS

As well as the choice of what part and type of the network to model, there is the choice of how to model it, i.e. the choice of programming language or modelling method to use. This choice is as varied as the choice of what to model, and often depends on the type of model.

A common theme is to use graph-theory or mathematical equations (including probability equations) for analytical type models [26], and either general purpose programming languages, such as C or C++, or specialised languages, such as ns2 [27], OpNET [28] or BONEs [29] for simulation models. It should be noted that the specialised languages mentioned here are themselves extensions to C or C++ and can often be used within a standard C/C++ program.

This split is due to the nature of analytical vs. simulation models. In simulation the focus is often the sequence of events (control) that occur during operation, for example the channel access procedure in a wireless network [21] and a programming language by its very nature is more suited to this type of modelling.

2.2.4 POWER LINE NETWORK MODELLING

As the work presented here is on modelling a HomePlug power line network, a review of the efforts made in modelling both HomePlug networks and power line networking in general is beneficial. This area isn't as active a research area as wireless networking however there has still been substantial work carried out.

Much of the work focuses on a single aspect of the system, such as the Media Access [30], data modulation [31] or the channel [32]. To date there hasn't been a unified model of a power line networking system presented. Further many of the models make assumptions that are unrealistic if considering a resource limited SoC based system. For example, in [33] the authors limit the data encoding to QPSK $\frac{3}{4}$ rate (although

HomePlug has many different encoding schemes) and in [34] the authors assume unlimited buffer for the storage of MAC frames.

Many authors propose extensions/alterations to HomePlug to improve the Quality of Service or Throughput of the protocol. In [35] and [36] two different research teams look at altering the Contention Window algorithm used in HomePlug (see section 3.3 for details of this) to obtain a higher throughput. The main differences in the methods are the way in which the results are obtained. In [36] the authors use an analytical model back up with a discrete-time simulation, whereas in [35] the authors use only a discrete-time model (developed using ns2) along with the power line channel model presented in [32]. The choice of channel model is odd however, as [32] is presenting a model of the local transformer/substation to the home (the “last-mile” of a power line network), however HomePlug is an in-home system which has different characteristics.

Other work looking at alterations to the MAC include [37] which looks at modifying the framing of the data to increase efficiency of transmission, [38] which looks at a new MAC level access scheme called Periodic Contention-Free Multiple Access and [39] which presents a new MAC entirely which the authors call HomeMAC. They take the slightly different approach of using an emulation system to develop it, and implement a PC based network (running FreeBSD) using their MAC but running on Ethernet as the physical communication link.

Further work has been done in [31] and [40] looking at extensions to the standard HomePlug Physical Layer. In [31] the authors look at an alternative to the HomePlug ROBO modulation scheme for poor channel conditions (see Section 3.x for more details of ROBO). In [40] the authors present an extension to the OFDM scheme used in HomePlug, namely Discrete Multi-Tone (DMT), which is a multi-carrier system similar to OFDM, but using a variable bit-rate encoder for the subcarriers (in OFDM all subcarriers are modulated using the same scheme, but in DMT different sub-carriers can be modulated with different schemes).

Some interesting work has been carried out in [30] and expanded upon in [34]. In these papers the authors look at the Throughput of the HomePlug MAC under both saturation conditions and normal operating conditions. They developed an analytical model of the

HomePlug MAC using a Markov model (a tri-dimensional discrete-time Markov Chain). The conclusion of the work is the throughput decreases as more stations are added.

2.2.5 NETWORK MODELLING SUMMARY

Although modelling networks is considered important, the area is so broad that there are many different methods and approaches to the problem. This in turn suggests that there is no single unified approach to modelling networks.

However, certain solutions are more suited to certain problems. For example, when modelling a specific aspect of a protocol, often analytical approaches are taken, and when modelling more than one aspect, or the interaction between components, often a simulation (or emulation) based approach is taken. This isn't always the case however, for example in [14], the author models the DCF of the 802.11 MAC using Markov Models, but in [27], the same function is modelled using ns2, which is a simulation based approach although in both cases the choice of how to model the network is unclear.

Much of the focus of modelling power line systems has been looking at one specific aspect of the protocols, such as the MAC throughput [30] or PHY encoding [40]. There are currently no models of the whole protocol and system. Many of the models that have been developed are analytical based, and don't place many real-world conditions on the simulation (such as limited processing power or memory availability).

2.3 HARDWARE MODELLING

2.3.1 OVERVIEW

As hardware has become more complex and the time-to-market has been reduced, the need for high-level models has grown [41]. Today it is almost unheard of to produce a complex SoC without a model with which to generate a set of golden reference figures. Models are also widely used to explore alternative solutions to problems [42].

Most hardware modelling languages are based around an object-orientated methodology such as SystemC [43, 44] or HASoC [45]. These have many advantages in hardware modelling as they appear to naturally match the nature of hardware [46]. However other languages have been used to model hardware – from dedicated hardware modelling languages such as Verilog and VHDL to higher level constructs such as data-flow models [47].

In this section, the various approaches to modelling hardware are described, along with some of the methodologies used. A useful point to raise at this time is levels of abstraction in hardware modelling. Ultimately, hardware is a series of connected transistors on a piece of silicon that produce a specific function or achieve a specific task. However, given that even the simplest designs are approaching millions of transistors, it is obvious that thinking of hardware at this level is not a good thing to do. Figure 2.4 shows the levels of abstraction in modelling hardware. The higher the level of abstraction, the simpler the design flow (for example at Behavioural Level, the designer is concerned with how the system works, but doesn't care what interactions are required to make it work).

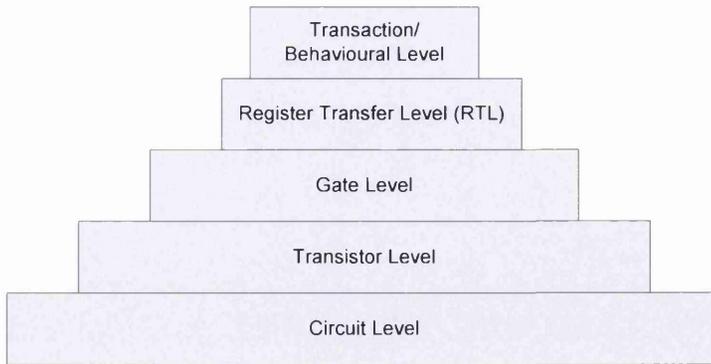


Figure 2.1 – Layers of Abstraction in Hardware Modelling [48]

An analogy that can be drawn between the hardware and software worlds is that of compilers. Behavioural models can be thought of as a high-level programming language, such as C, with the RTL level being the machine code that it gets compiled to.

2.3.2 LANGUAGES

As mentioned above, most hardware modelling languages use C [49, 50] or C++ [51, 52], an extension/derivation of these (SystemC [43, 44], SpecC [53, 54, 55], HandelC [56], etc.), or use the hardware description languages (HDLs) VHDL or Verilog [57]. Using HDLs, whilst it might sound like the best approach as the final hardware will more than likely be in that language, isn't always the best approach, as they don't offer the wide range of programming constructs that general purpose languages do [58], although this is being addressed with the advent of System Verilog [59] for example.

In [49], the author uses C to develop an MPEG-2 Decoder Intellectual Property (IP) Block. Whilst this isn't directly associated with networks, the design of IP blocks requires the same basic approach regardless of the end application. One of the reasons for using C (or indeed any of the languages described here) is the reduction in simulation time compared to the HDL implementation. Another advantage identified by the author of [49] is the fact that a non-hardware model can be used, which they term a "C-Soft" model, to verify the hardware model (in both C and Verilog). In their case, the hardware C model had the same hierarchy and function as the final Verilog model. The main benefit the authors found was the ability to verify the hardware block during simulation. This was done by comparing the outputs of the C hardware and Verilog models with the C software model.

Along with C based approaches to modelling hardware, C++ has also proved popular (or more exactly, object-orientated approaches [60] which are invariably implemented in C++). In [61] the authors describe an implementation of an Orthogonal Frequency Division Multiplexing (OFDM) based Wireless LAN (WLAN) transceiver, based on a C++ model and implemented in $0.18\mu\text{m}$ CMOS. They stress the importance of being able to go easily from the high-level C++ model to a register transfer level (RTL) model from which the design/model can be realised in hardware, as well as the need to be able to explore different architectural decisions. Another important point that is raised is the need to convert the initial floating-point model to a fixed-point representation, and the need to be able to explore different fixed-point representations. They proposed the design flow in Figure 2.5.

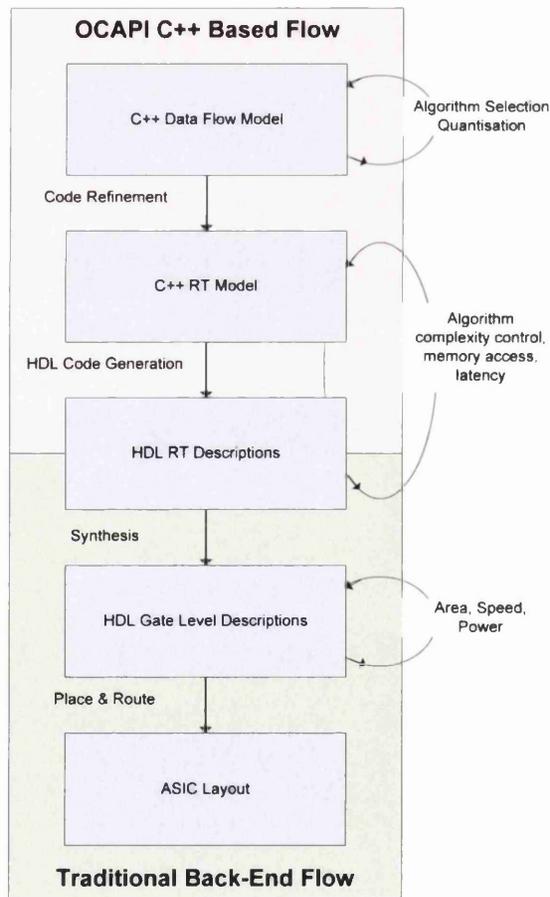


Figure 2.2 – C++ Based Hardware Design Flow [61]

Another paper using C++ as a modelling language is [62]. In it the authors point out that C/C++ as it stands is unable to model hardware specific concepts, such as concurrency and reactivity and needs an extension to be able to cope with this. They also propose that if the design doesn't need to be translated from C/C++ to HDL (prior to hardware synthesis), then there is less chance of error and the systems engineer can be involved much later in the design flow. The main approach is to abstract the hardware issues as much as possible and have a behavioural model that can be mapped directly to hardware.

Extensions to C and C++ are proving extremely popular as methods to model hardware. Perhaps one of the most popular at the time of writing is SystemC [63] and there have been many research papers on this [43, 44]. SystemC is a C++ class library that provides extensions to C++ that support hardware interaction/modelling.

In [64], the authors describe using SystemC to go from a high-level description down to a gate-level representation. At the time of the paper, there were no direct approaches to convert from SystemC to hardware, unless a subset of C/C++ was used that removed a lot of the high-level features (such as pointers and object orientation) which made using C++/SystemC attractive. To date there are still not any direct conversion/synthesis tools that go from SystemC to gate-level. The authors go on to describe an extension to SystemC that attempts to solve this problem, however it requires yet another tool (ODETTE) and uses object oriented HDL's, which are not widely used in industry.

SystemC has been used to model many different types of hardware system. In [65], the authors use SystemC to model and verify a simple on-chip bus, similar to the AMBA AHB. In [66], the authors use SystemC to develop a model of an existing ASIC (a networking chip), for the purposes of comparing the simulation time with that of the Verilog at RTL level, as well as attempting to trace a problem with the speed at which the chip was transmitting. They found that the SystemC model ran sufficiently fast to allow them to explore the problem they had with the chip. In [67], the authors use SystemC to develop a model of a SoC running a version of embedded Linux, and found the speed up compared to RTL simulations to be significant, reportedly up to 10,000 times faster.

An extension to C proposed for hardware modelling is SpecC [68], and this has been used widely as a teaching tool [69]. SpecC allows the same semantics and syntax to be used to represent a design at the various stages through the design flow [70]. It has been used to model various devices, such as asynchronous circuits [71], multi-processor SoCs [72] and real-time emulators of electromechanical systems [73].

The final main approach to modelling hardware is to use the HDL's themselves. Although these offer many of the higher level constructs of a general purpose language, such as for loops, these are generally not used when the HDL is used for RTL descriptions of the hardware. However they still do not offer the flexibility that a general purpose language does. Using HDLs does mean that a single language can be used to describe the model and describe the RTL description to the final place-and-routed gate-level description. To this end, the IEEE is working on the next version of the Verilog HDL (Verilog-2005) to include SystemVerilog and address some of these issues.

2.3.3 SIMULATION ISSUES

Along with the issue of how to model the hardware, there is also the issue of how to simulate the hardware model. If the model is in an HDL, then it can be simulated using one of the many HDL simulator tools (such as Cadence's Verilog-XL or Mentor's ModelSim). If the model is implemented in a high-level language, then the issue of how to best model the hardware as though *it is actually* hardware becomes an issue.

In [74], the authors discuss methods of computation for embedded systems. They state that concurrency (i.e. multiple components running simultaneously) must be considered at all levels of the model. This is essential, as one of the fundamental properties of hardware is its concurrent nature. This can also be termed reactivity, i.e. hardware reacts to events that occur, generally outwith its control. This is a concept backed up by [75].

As well as any simulation model requiring concurrency, the authors of [74] also state that the components that make up the model need to be able to communicate with each other. This is intertwined with their concept of Models of Computation (MOC), and the most relevant one they suggest is Discrete Event. In this model, communication is achieved by multiple-writer, single-reader signals/messages that carry information as well as time. This method they say, is well suited to simulation as it is the only one that

represents time. This is the model used by logic simulators. Other MOCs proposed include; Data Flow networks where data is passed from one process to another in zero time; Petri Nets where data is ordered over transitions on graphs or nets; and Synchronous State Machines, where events occur at discrete “clock” events (time instants).

In [76], the authors give some of their key requirements for a hardware simulator:

1. Precision The model needs to be sufficiently accurate in describing the hardware (although “sufficiently” depends on the application)
2. Efficiency The model must simulate as quickly as possible
3. Separation of Events The simulation will likely only want to emphasise certain aspects of the model under test/simulation
4. Flexibility The model must be easily adapted and extended

The sort of models that the authors of [76] are talking about are macro cell models (which are gate-level models of transistors to replace SPICE models), however the points that they making are perfectly valid for higher-level abstraction models, such as those considered here.

A final point about the simulation of the model is raised in [77], where the authors describe what they call a hardware modeller, although a more accurate description would be hardware emulator. They do however raise one important point, which is any modelling software, needs to be portable across various implementation platforms.

2.3.4 SUMMARY OF HARDWARE MODELLING/SIMULATION

In this section a brief walkthrough of hardware modelling and simulation has been presented. This is still a very active research area, especially as the complexity of hardware increases. The section introduced some of the important concepts of modelling, such as the level of abstraction that the model takes. The more abstract the model the quicker it will simulate, however there is a risk of not modelling important

aspects of the hardware device in question. As with most engineering problems, there is a trade off between how accurate the model is and how quickly the engineer wants it to run.

Also introduced are the various languages used to model hardware. These are almost as varied as those used to model networks; however a common theme here is the use of high-level (general purpose) languages such as C or C++ to model the hardware. There is a caveat to this, in that language extensions are required to support the “hardware” aspects of the model, such as the concurrent nature of any hardware system.

This brings into play the other important point raised, that is how to simulate the hardware models. As hardware is concurrent “processes” that communicate with each other, a method is needed to model this. It already exists in HDL simulators; however these are too low an abstraction level for the work considered. The method to provide the functionality required is known as Discrete Event simulation, in which events are passed between the concurrent processes to enable functions or tasks to be performed.

2.4 SUMMARY

In this chapter a joint look has been taken at both network and hardware modelling. Network modelling has been around for much longer (since the first networks were developed) and many different approaches have been proposed over the years. Hardware modelling hasn't been around quite as long (probably 20 years or so), but there are many different approaches to this as well. A common theme in both is to use a high-level language such as C or C++ to model the network or hardware. This provides many advantages, such as speed of simulation, ease of development and abstraction away from some of the complex issues of realising a design in an ASIC.

In order for C/C++ to be used in simulation, which is one of the most common methods of modelling multi-protocol level network models, extensions need to be developed to provide the elements necessary to model the concurrent, reactive nature inherent to hardware. Several hardware modelling extensions have been presented such as SystemC and Spec, however as long as the model developed shows concurrency and uses a discrete event simulator, the ability to simulate the network/hardware will be met.

The ideas presented in this chapter are the basis of the requirements of the network model presented in Chapter 4. In that chapter a lot of the abstract ideas introduced in this chapter are expanded to provide a working network/hardware model.

Chapter 3 - Background Theory

Provides a more detailed description of the theory and standards that were used in creating the system model. It starts with a brief history of computer networking and highlights some of the relevant standards. The HomePlug power-line standard is then described in some detail, before a brief overview of the channel model is given.

3.1 INTRODUCTION

From the discussions in the previous chapter, it can be seen that a model that includes all aspects of a communication system is required, not just specific components. To validate this idea, the system shown in Figure 3.1 was developed. This is built around the HomePlug Power-line networking standard, and includes an abstract version of the higher level components, along with the Media Access Controller (MAC), Physical Layer (PHY) and channel. This chapter contains a description of these components.

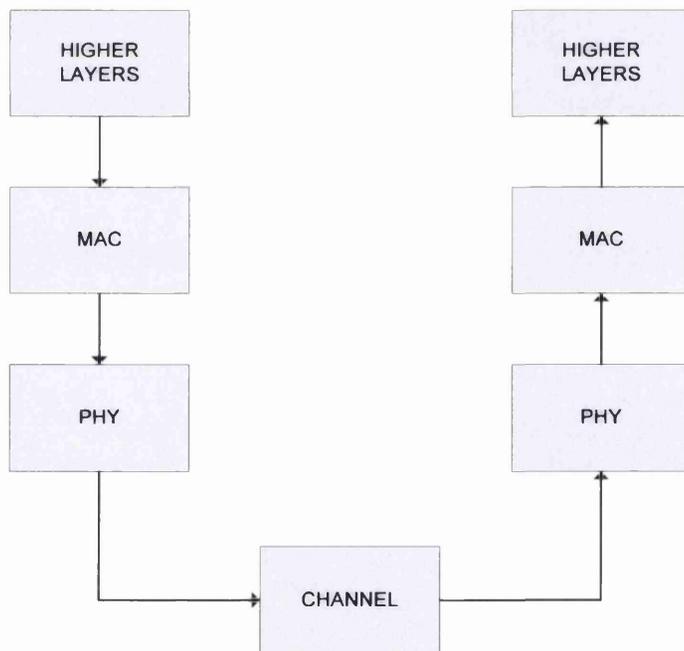


Figure 3.1 - Basic Model Block Diagram

3.2 COMPUTER NETWORKING

3.2.1 HISTORY

In the 1970s the use of computers in business and industrial applications started to dramatically increase, however the cost of the equipment was still very high. At this time, various people realised the advantage of connecting the computers together, which meant that expensive resources such as processors and printers could be shared along with data. Initially various companies (such as IBM and Xerox) developed their own proprietary networking standards that only enabled their equipment to be connected. This was a far from ideal situation, as it meant that a company was tied to one brand for all its networking equipment, and if they chose the wrong one it could have consequences for the business. This issue was dealt with by the standard bodies developing industry-wide standards describing how computers should communicate. The major networking protocols were standardised in the late 1970s by the 802 committees of the Institute for Electrical and Electronic Engineers (IEEE). The IEEE Standards are based on the International Organisation for Standardisation (ISO) Open Systems Interconnection (OSI) 7-layer network model. They do however concentrate on the lower layers of the ISO-OSI model which are the ones associated with the actual physical medium that is used to transmit the data, rather than the higher layers, which give meaning and structure to the data.

3.2.2 OSI MODEL

The OSI network model, which was released in 1979, gives an abstract description of how two devices communicate with each other using a hierarchical layered model. Each layer provides services to the layer above, and uses the services of the layer below. In this way, data from any application can be transmitted over any physical network. Table 3.1 summarises the function of each of the seven layers in the OSI model.

Layer	Function
Application	Provides the interface between the communications environment and the applications using it. Example applications include electronic mail, web browsers, distributed databases and network operating systems.
Presentation	Provides services to application layer such as file transfer, virtual Terminal, text compression and encryption.
Session	Starts to add meaning to the data and provides basic user orientated services. It also provides the mechanism for controlling the dialogue between presentation entities.

Layer	Function
Transport	Provides the mechanisms for the initial establishment of communications channel, the transfer of data and the final release of the channel. It also interfaces with the Network layer to ensure an error free virtual point-to-point connection.
Network	Provides rules for routing in the network, and ensures not too many packets are in the system. It is also responsible for network addressing and call set-up and clearing.
Data Link	Ensures reliable communication over the physical layer, and provides the structure of the data (i.e. the frame format), along with the rules for accessing the channel. It also provides flow control and error correction and recovery.
Physical	Defines the physical (electrical and mechanical) attributes of the network, including modulation scheme, cable sizes, connector types, etc. It provides the means to transmit data across the network medium.

Table 3.1 - Description of OSI Layers

When two devices are communicating, each layer thinks it has a direct link with the corresponding layer of the other device, however in reality the only layers in direct contact are the Physical layers. Figure 3.2 shows how one device communicates with another (with an intermediate bridge/repeater which allows the two devices to have different physical layers)

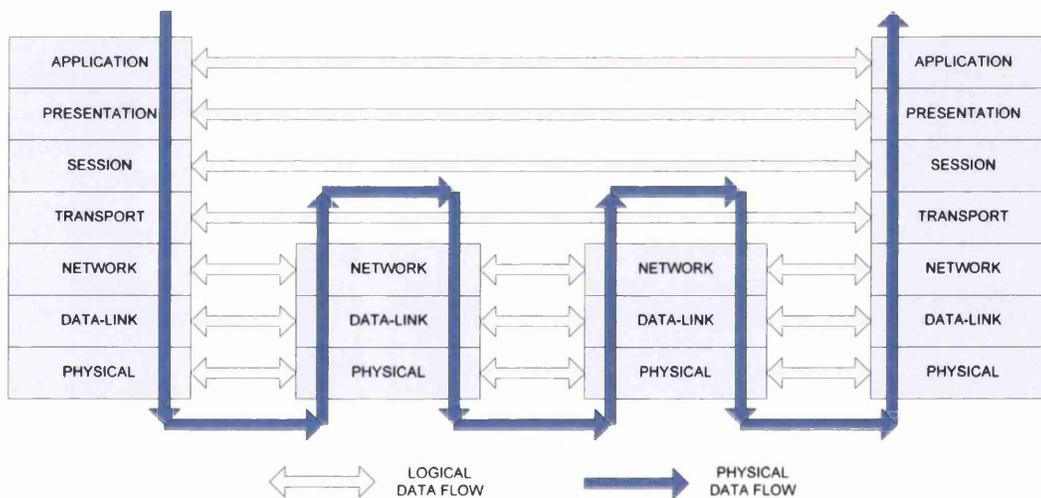


Figure 3.2 - Two Devices Communicating Using the OSI 7-Layer Model

As the data passes down the layers from the Application Layer, extra control and routing information is added that is necessary for the correct operation of that layer. This process is known as encapsulation, and is shown in Figure 3.3. The sort of information added includes IP Addresses, CRC checksums, protocol versions, etc.

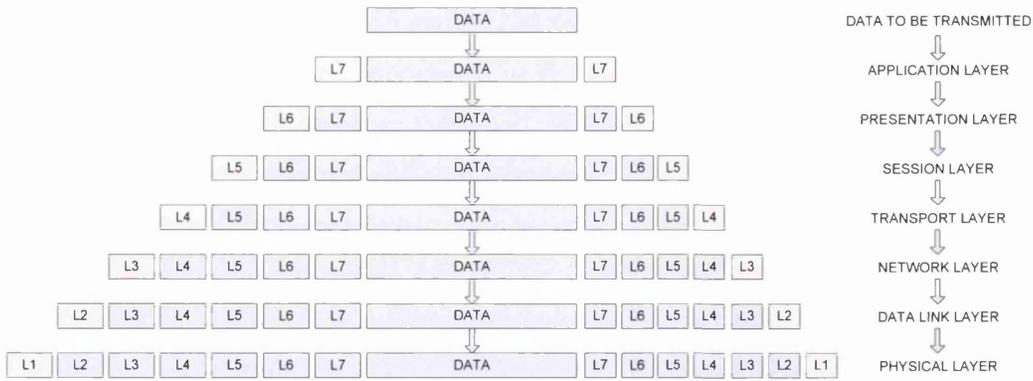


Figure 3.3 - Data Encapsulation

The OSI model describes an abstract networking model and protocols have been developed and refined to fit this model. These range from physical network protocols (such as Ethernet and wireless), through transportation protocols (TCP/IP, NetBIOS) to application protocols (HTTP, FTP, etc.). The relationship between these protocols and the OSI model is shown in Figure 3.4.

APPLICATION LAYER	HTTP, SMTP, FTP, etc.		
PRESENTATION LAYER	XML, SMB		
SESSION LAYER	SSH		
TRANSPORT LAYER	TCP, UDP		
NETWORK LAYER	IP, ICMP, ARP		
DATA-LINK LAYER	ETHERNET	TOKEN RING	WIRELESS
PHYSICAL LAYER			

Figure 3.4 - Network Protocol Stack

3.2.3 IEEE NETWORKING STANDARDS

Most of the work standardising the physical network protocols has been done by the various IEEE 802 sub-committees. They have been primarily concerned with the bottom two layers (Data-Link and Physical) and have developed many different types of network, the most common today being Ethernet (802.3) and Wireless (802.11), although the work being carried out in Personal Area Networks (PANs) and Metropolitan Area

Networks (MANs) by the 802.15 and 80.216 sub-committees to create more ubiquitous networks might have more importance in the future. The more common network protocols developed are given in Table 3.2. Work is ongoing to improve and enhance these standards to exploit technological advances. This can be seen in the evolution of the 802.3 (Ethernet) standard, from a 2Mb/s system, through 10Mb/s and 100Mb/s to a 1GB/s system.

Sub-Group	Standard
802.1	LAN/MAN Management, Bridging
802.2	Logical Link Control
802.3	Ethernet (CSMA/CD)
802.4	Token Bus
802.5	Token Ring
802.6	Distributed Queue Dual Bus
802.10	Interoperable LAN/MAN Security
802.11	Wireless LAN
802.15	Wireless PAN
802.16	Wireless MAN

Table 3.2 - IEEE 802 Sub-groups

The IEEE split the Data-link layer into two sub-layers, the Logical Link Control (LLC) layer and Media Access Control (MAC) layer. This provides a standard interface to the higher layers through the LLC for any MAC/PHY that is developed. The MAC is the part of the protocol that controls access to the network and ensures each device follows the correct procedure for gaining control of the physical medium. It also provides the last level of framing before the binary data is modulated by the Physical (PHY) layer, as well as possibly segmenting the incoming message if it is larger than the maximum permitted message size on the PHY. A maximum PHY frame size is often imposed to allow fair access to the medium. Generally, the MAC/PHY comes as a pair, as the framing and access mechanisms will be unique to the specific PHY being used, although there are some exceptions, notably the Home Phone-Line Network Alliance (HPNA) PHY uses a standard Ethernet MAC. This was done to allow easy development of HPNA products by companies that already had Ethernet products.

3.3 HOMEPLUG

3.3.1 OVERVIEW

The HomePlug Consortium [7] was established in the late 1990s by leading communication companies to develop a robust, high-speed data-networking standard for the power-line. The companies involved realised that unless they worked together then each could develop a proprietary standard that wouldn't work with other products, similar to the situation that arose in the early days of data networks. Some of the founding members include Intellon (who provided much of the technology for the subsequent standard), Conexant, Panasonic and Sharp. There are currently over 40 members.

Version 1.0 of the standard was released in June 2001 [78], and has a raw data rate of 14Mb/s (so roughly equivalent to early Ethernet). Due to the way the data is transmitted on the channel this rate is not guaranteed, and it will in fact adapt to the conditions of the channel. A slower, more robust modulation will be used on poorer channels. Extensive field trials have been conducted in the United States (however the results are confidential), as many of the products are only available there as it is a very large market, plus many of the companies involved are American. One of the reasons adoption of the technology in Europe has been slower is the different electricity supply, which is 240V, 50Hz single-phase in Europe and 120V, 60Hz two-phase in the US. There is also a regulatory issue, as the European Telecommunications Standards Institute (ETSI – the European equivalent of the Federal Communications Commission, FCC) have specified the frequency ranges to be used for power-line communications, and have split these into “Access” and “In-Home”. “Access” is using the power-line entering the house to carry broadband traffic and “In-home” is networking internal to the house (i.e. a power-line LAN such as HomePlug).

Unfortunately, this split is in the middle of the frequency range used by HomePlug, and so this means that the maximum raw data rate will be lower (about half, or around 7Mb/s). Figure 3.5 shows the ETSI-specified frequencies, along with those used by HomePlug. There is however, a second version of the regulations that should overcome this problem, and allow use of the full frequency range, as long as there are no “Access” services present [79].

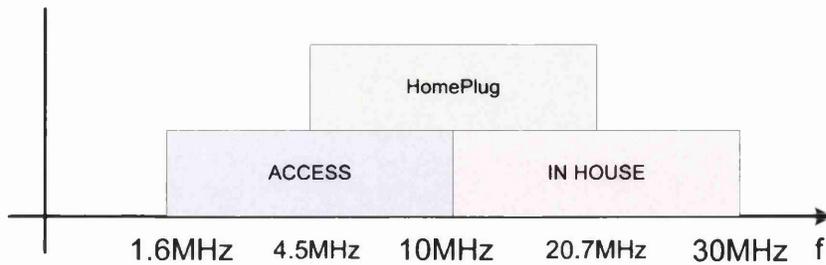


Figure 3.5 - ETSI & HomePlug Power-line Frequency Ranges

The standard specifies both the Media Access (MAC) and Physical (PHY) layers and these are similar to the MAC and PHY used by the wireless networking standard (802.11). The reason for this is that the power-line exhibits much of the same characteristics as the wireless channel in that it has fading multi-path channels (see Section 3.3), and is a very noisy and hostile environment through which to transmit data. The MAC uses a Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) access scheme, with a four-level priority scheme and frame acknowledgements to improve Quality of Service (QoS). The PHY uses Orthogonal Frequency Division Multiplexing (OFDM) with Forward Error Correction (FEC) to increase the probability that the frame is received error free. OFDM is a form of multi-carrier modulation that uses many (equally spaced) sub frequencies over a given bandwidth and each sub-carrier is modulated separately. In HomePlug's case, there are 84 carriers between approximately 4 and 20 MHz, and each can carry 2, 1, or $\frac{1}{4}$ bits depending on the modulation scheme used.

Security is an issue, with HomePlug using encryption to ensure that adjacent homes cannot read each others data. This creates logical networks over the shared physical medium and is shown in Figure 3.6. Each HomePlug station must be able to store at least one encryption key; however, it is feasible for the station to store more than one key, thereby letting it operate on more than one logical network.

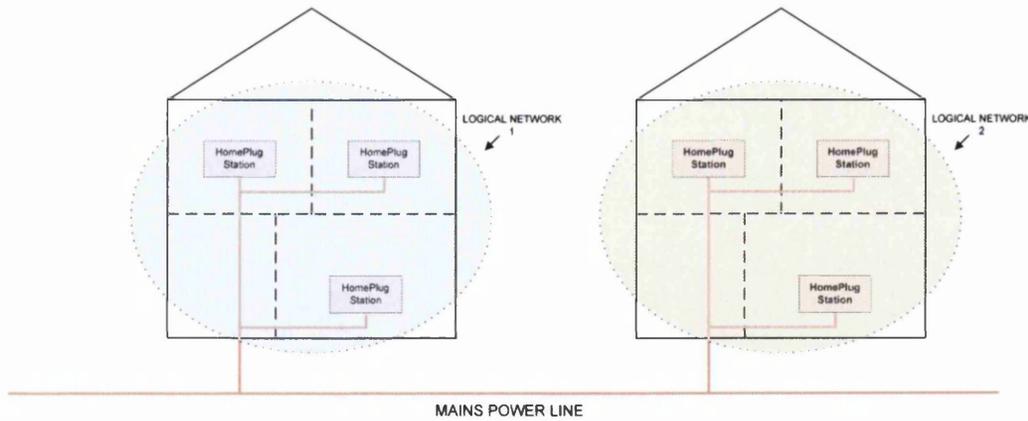


Figure 3.6 - HomePlug Logical Network

3.3.2 HOMEPLUG MAC

As stated, the HomePlug MAC is similar to the 802.11 MAC, but with some enhancements to improve the Quality-of-Service (QoS). The MAC also uses message segmentation to support the rate adaptive nature of the PHY. As the physical channel exhibits many properties similar to the wireless channel (i.e. fading multi-path), each node must perform channel estimation prior to transmission to determine the PHY modulation parameters. Each node's MAC maintains a list of the usable carriers, modulation technique and convolutional code rate for transmission to various destination nodes. The usable carriers, modulation technique and convolutional code rate are referred to as the Tone Map, and the list is called the Tone Map Index. HomePlug also uses an acknowledgement scheme (i.e. Automatic Repeat Request or ARQ) to ensure frames get delivered, although it isn't compulsory. The acknowledgement can either be positive (ACK) or negative (NACK or FAIL). A NACK response indicates that the frame was received, but with errors and a FAIL response indicates the receiving node doesn't have the resources to decode the frame. HomePlug also implements a "Partial ARQ" scheme for multicast frames, in which a single station acknowledges the reception of a frame for all the nodes on the network.

The MAC is capable of transporting frames between 46 and 1500 bytes in length, and so can transport Ethernet frames by encapsulating them within HomePlug frames. This allows easy bridging between HomePlug and other networks. Standard IEEE 48-bit addressing is used at the MAC level.

HomePlug uses a four-level priority scheme and before transmission all the nodes on the network will perform Priority Resolution to determine the priority of the traffic that is transmitted on the network. This stops lower priority traffic from congesting the network. If a node has multiple segments to transmit (to a single destination) it can use Segment Bursting, which means that once it has gained access to the network it can transmit the complete segment without releasing control, unless higher priority traffic appears. This idea is extended for a node that has multiple highest priority frames to send (not necessarily all to the same destination) in Contention Free Access. This allows the node, once it has the channel, to transmit up to seven consecutive frames without needing to perform the normal channel access procedures.

Encryption is provided using a 56-bit Data Encryption Standard (DES) scheme, and this is done before the frame is segmented and transmitted. Details of what is encrypted are given in Section 3.3.2.1.

3.3.2.1 Frame Assembly

The frame assembly process in HomePlug is more complicated than that of Ethernet due to the nature of the PHY encoding scheme. This is because the amount of data that can be transmitted in a maximum length PHY frame is variable, and the MAC needs to take account of this when generating the frames for transmission. The sequence of events is shown in Figure 3.7, and described in the paragraphs following.

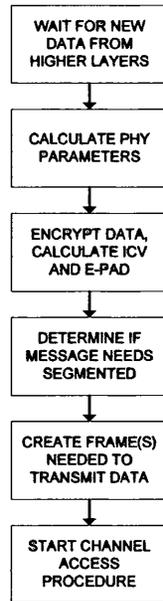


Figure 3.7 – Frame Assembly Sequence

Before the MAC begins the frame assemble process, it needs to know the parameters used by the PHY during encoding, namely the modulation technique, number of usable carriers and the convolutional code rate. This information is held in the Tone Map, which is obtained by the nodes involved in the transfer performing channel estimation, prior to data transmission to determine the fastest stable transfer rate and is a MAC management function (see Section 3.3.2.3).

When the MAC gets a request to send data (called a MAC Service Data Unit or MSDU), it begins the segmentation and PHY frame assembly process. Segmentation might be necessary if the MSDU (plus MAC level information) is larger than the maximum length PHY frame. The MSDU consists of the payload (the data from the higher layers), an optional Virtual LAN (VLAN) tag for 802.8 compatibility and a Type/Length field, along with the source and destination MAC addresses. The addresses are used to determine the status of the Tone Map, and are present in each frame transmitted. If the Tone Map is invalid, then the MAC will send a Channel Estimation MAC Management frame (see section 3.3.2.3 for information on MAC Management data).

Once the MAC has the information from the Tone Map, it will begin to assemble the frame(s) needed to transmit the MSDU (also known as MAC Protocol Data Units or MPDU). The first part of this is to encrypt the MSDU (for the logical networks), and

generate the Encryption Control, Encryption Pad (E-PAD) and Integrity Check Value (ICV) fields. The Encryption Control field contains the 1-byte Encryption Key Select field (which is the key used to encrypt the data) and the 8-byte Initialisation Vector field (which is the “starting” value of the encryption engine). These are needed by the receiver to properly decrypt the MSDU. Because the encryption engine operates on data in blocks of 64-bits, the E-PAD field is used to ensure that the MSDU is a multiple of 64-bits. The ICV is a 32-bit CRC that is calculated over the bits from the start of the Encryption control field to the end of the E-PAD. This gives the “Service Block” which is then segmented into the frames needed before being transmitted on the channel. The structure of this is shown in Figure 3.8 along with those fields that are encrypted and those that are used to calculate the ICV. It also gives the size (in bytes) of the various fields.

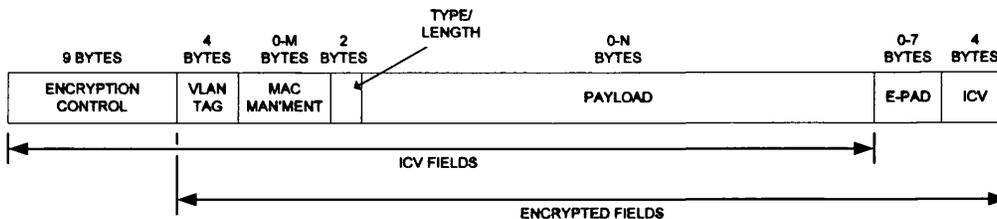


Figure 3.8 - Service Block Structure

Before the Service Block is segmented, the MAC has to calculate the number of bytes that can be transmitted in a PHY block (or more specifically, the number of bits that are transmitted in each PHY symbol). The HomePlug PHY transmits blocks of 20 and 40 symbols¹ (up to a maximum of four 40-Symbol blocks or 120 symbols). Once these calculations have been performed, the number of frames required to transmit the Service Block is determined. There is an additional overhead of 19 bytes (composed of the frame header and the CRC) for each frame transmitted, which the MAC must take into account when doing the segmentation.

Once the MAC has determined the number of frames that are needed to send the service block, it can begin to segment the message and generate the other fields that are needed to create the complete frame. These are the Segment Control, Destination Address,

¹ A symbol is the unit of data that is transmitted on the PHY. It consists of multiple bits, depending on the parameters used in the PHY, up to a maximum of 168 bits for HomePlug.

Source Address, Payload, Block Pad (B-PAD – although this is only in the last frame of the Service Block) and Frame Check Sequence (FCS). Figure 3.9 shows the layout of these fields (which is also the layout of the frame that is sent to the PHY).

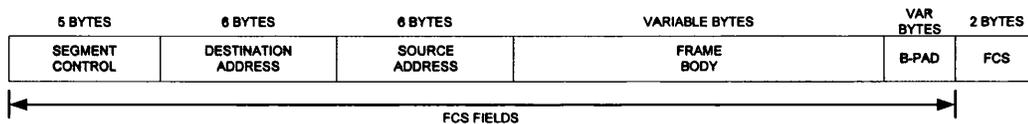


Figure 3.9 - MAC Frame Structure

The Segment Control field is a 5-byte long field which contains the information needed for the receiver to re-assemble the frame, and Table 3.3 shows the sub-components that make up this field, their sizes and what they are for. The Destination and Source addresses are standard 48-bit IEEE MAC addresses and are given to the MAC when it is requested to send an MSDU. Payload is a variable field containing the portion of the Service Block that is being transmitted in this frame. B-PAD is only present in the last (or only) frame of a segmented message, and is required to ensure that the data out of the MAC is of sufficient length to fill a complete 40 or 20 symbol PHY block. The size of this is calculated at the same time as the parameters for the segmentation. The FCS is a 16-bit CRC that is used to determine if the frame has been received error free.

Field	Definition	Byte	Bits	Description
FPV	Frame Protocol Version	0	7-5	Indicates the protocol version. Set to 000
RSVD	Reserved	0	3-4	Reserved. Set to 00
MCF	Multicast Flag	0	2	Indicates the MPDU contains a multicast payload and the destination address indicates a unicast address. This allows the Partial ARQ to be used
CAP	Channel Access Priority	0	1-0	The priority of the message. It is repeated in the End Frame Control
SL	Segment Length	1 2	7-0 7-1	The number of bytes in the Frame body, exclusive of the control information
LSF	Last Segment Flag	2	0	Indicates current segment is last or only segment of the Service Block
SC	Segment Count	3	7-2	The number of the current segment. It is incremented for each segment of a Service Block, and is used by the re-assembly process
SN	Sequence Number	3 4	1-0 7-0	Identification number for each new Service Block that is transmitted.

Table 3.3 - Segment Control Field Structure

The final part of the frame that needs to be generated is the start and end delimiters. These consist of a preamble which allows the receiver PHYs to “wake-up” and to

determine the channel conditions (although the preamble is in fact generated by the PHY) and a Frame Control field. The Frame Control field is a 25-bit field with the structure shown in Figure 3.10. It is encoded in the PHY as a 4-symbol block.

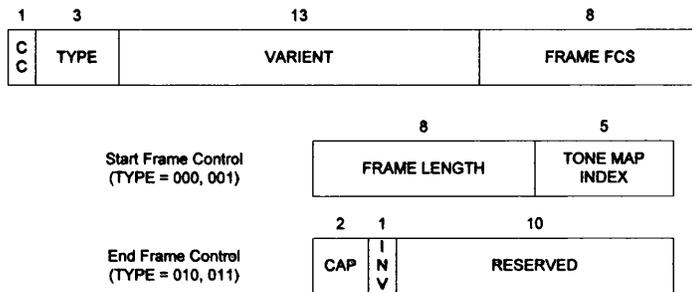


Figure 3.10 - Frame Control Structure

The Start Frame Control gives the receiver the length of the frame (in terms of PHY blocks) and the Tone Map index. The length of the frame gives the nodes on the network the information needed to determine the time for the Virtual Carrier Sense (VCS) timer (along with the frame type, as this indicates if a response frame is needed). The Tone Map Index allows the receiver to select the correct Tone Map to allow the PHY to decode the incoming message. The CC field (present in all frame control) indicates if the message is part of a Segment Burst or Contention Free access. The Frame FCS (FFCS) is an 8-bit CRC used to determine if the Frame Control was received error free. The End Frame Control contains the priority of the message (CAP) and an invalid flag, which is set to 0 on transmit and if it is received as a 1 then the Frame Control is invalid (although this would probably be picked up by the FFCS). The rest of the End Frame Control is reserved, and is set to all 0's. Figure 3.11 shows the frame assembly process from the data received by the MAC to the data passed to the PHY.

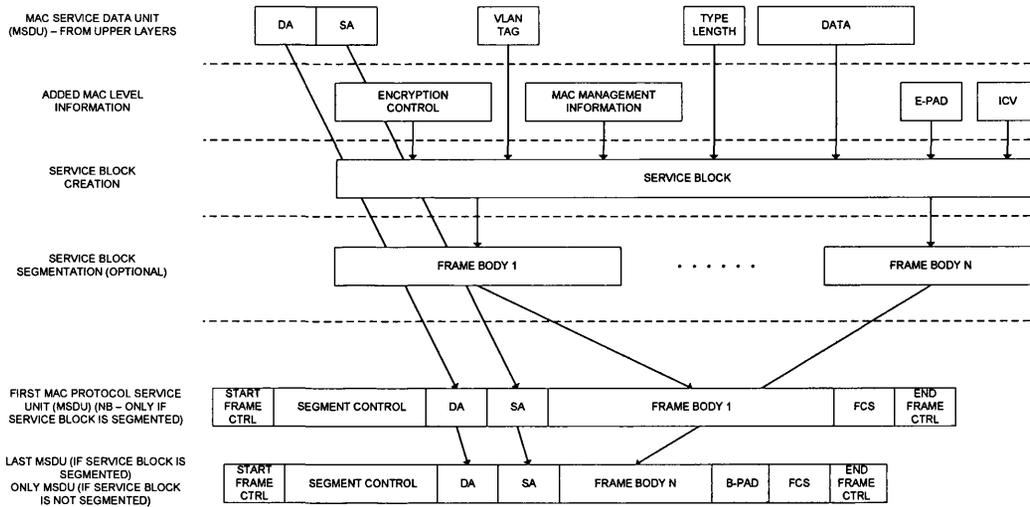


Figure 3.11 – Segmentation Process

At the receiver side, the process is reversed. It will receive each frame from the PHY, and determine if the frame is error free by checking the received FCS against the locally calculated version, and checking both the start and end frame delimiters. There are also certain fields within the delimiters and frame header that are reserved and these must be the correct value. If they are incorrect or the FCS is invalid, then the frame is in error. Once the receiver has determined the status of the frame, it will generate the appropriate response frame. This can be one of three; Positive Acknowledgement (ACK), Negative Acknowledgement (NACK) or Fail (FAIL). ACK indicates that the frame was received without error, NACK indicates the frame was received with error and needs to be retransmitted, and FAIL indicates that the receiver does not have sufficient resources to process the frame (for example it has received a lot of frames, and its buffers are full). The response frames are the same as the start and end delimiters (i.e. they are made up of Preamble and a Frame Control), however with the appropriate value in the type field. The format is shown in Figure 3.12.

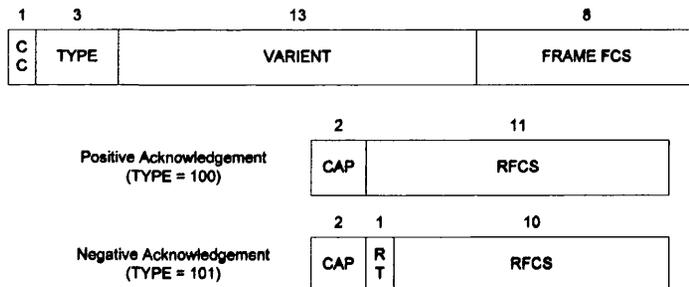


Figure 3.12 - Response Frame Control

The “CAP” field is a copy of the “CAP” field in the Segment Control block and gives the priority of the transmitted message, “RT” is the Response Type and indicates if the negative acknowledgement in an ACK (RT=0) or a FAIL (RT=1). “RFCS” is the received Frame Check Sequence from the transmitted frame, and is used by the node sending the original frame to ensure the response is for that frame.

They are transmitted at a specific time after the end of the received frame (the Response Interframe Space or RIFS), as shown in Figure 3.13. Other nodes on the network know if the frame that has just been transmitted will require a response (from the Frame Type in the Start and End Frame Control), and will not begin the Channel Access process until a Contention Interframe Space (CIFS) has passed. The diagram gives two views of the process. The upper view shows time progressing down the diagram, and the flow of data between the two nodes involved in the transmission. The second view shows the data as it appears on the physical network (with time progressing left to right).

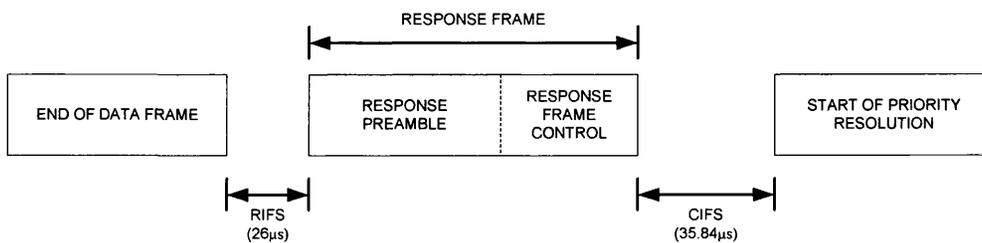
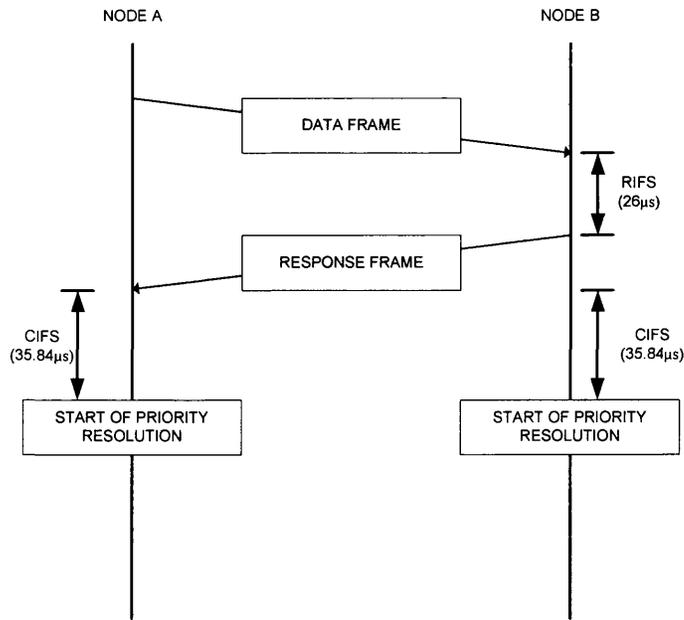


Figure 3.13 - Response Frame Timing

3.3.2.2 Channel Access

As mentioned above the MAC's other function is channel access. This ensures all stations access the channel in the proper manner and at the proper time. It also ensures that access is fair. The basic access mechanism is Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA). The MAC uses two types of carrier sense to determine when the physical network is busy, Virtual Carrier Sense (VCS) and Physical Carrier Sense (PCS). VCS is maintained by the MAC and is updated depending on the control information that is received by the node. This uses the information in the Start and End Delimiters to determine the time that the channel will be busy and the delimiter type to determine what the next thing on the network will be (i.e. the start of the channel access or a response frame).

Once the MAC has a properly formatted frame to transmit, which will consist of a Start Delimiter, Payload and End Delimiter, it will begin channel access. The flow chart in Figure 3.14 shows the process pictorially, and it is described in the next paragraphs.

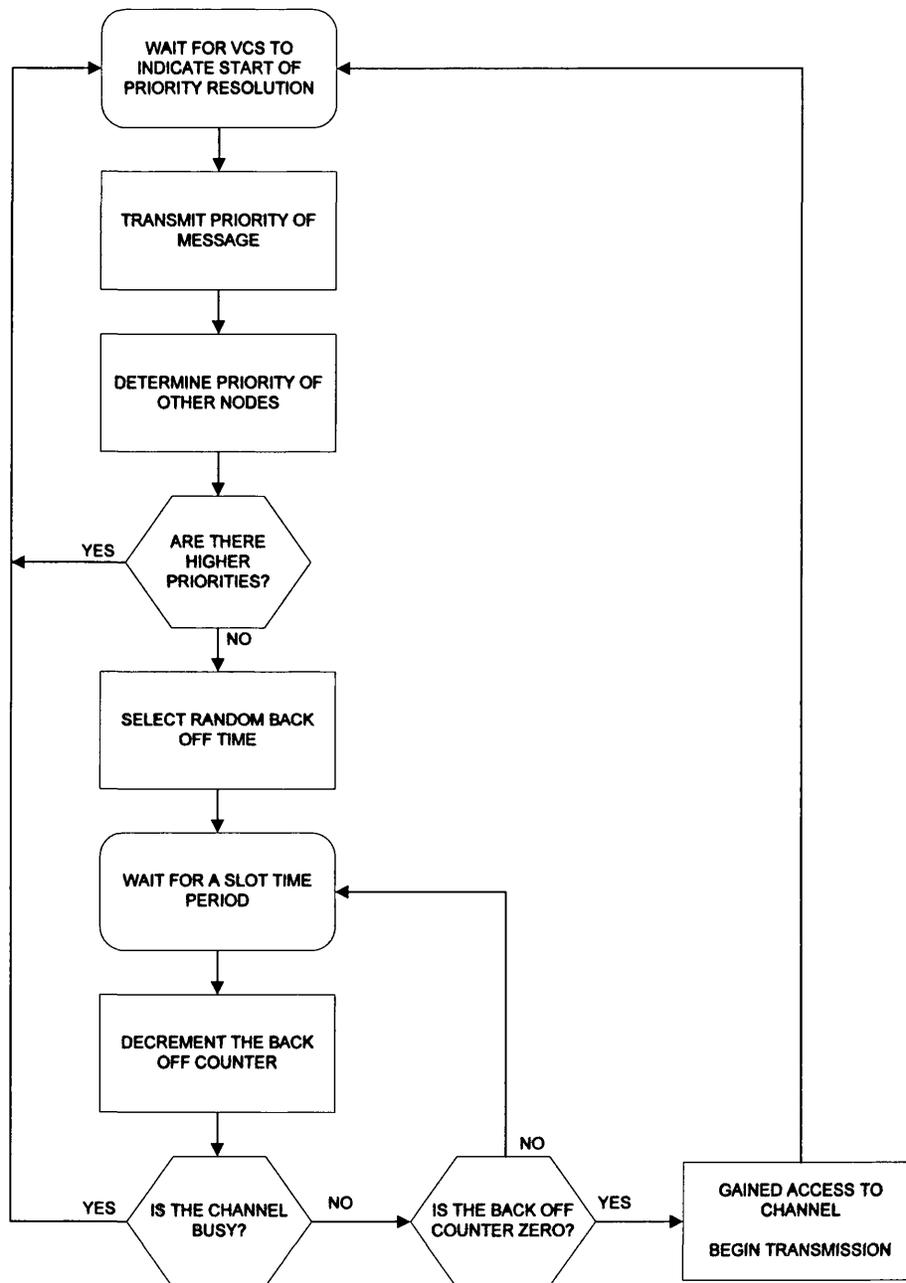


Figure 3.14 - Channel Access Process

The first step is to wait for the VCS timer to expire which indicates that the previous frame (and any response) has been transmitted. At this point, the priority resolution period can begin. Each node on the network should begin this at the same time, as they

will have been monitoring the traffic on the network and will have set their VCS timers accordingly. At the first Priority Resolution Symbol (PRS) symbol time, the stations will transmit their first priority symbol and at the same time will receive the PRS symbols transmitted by all the other nodes. Due to the way the symbols are modulated a PRS1 will “over-write” a PRS0, and so if a station transmits a 0 and receives a 1 it will know it has lost the priority resolution, and will not transmit its second PRS symbol. Those nodes that haven’t lost the first PRS symbol will transmit the second one at the correct time, and again will receive what the other nodes have transmitted. If the node doesn’t lose the second PRS symbol, it will begin the contention period, which is needed because there could be more than one node with the same priority.

At the start of the contention period, each node will choose a random value that is within the contention window for that transmission (if this is the first attempt to gain channel access for the frame) or continue with the value it had when it lost contention the last time. The size of the contention window depends on the priority of the message, plus the number of times that the node has attempted to transmit the message and it has failed (i.e. the node has actually transmitted the message, but the receiving station has returned a NACK or FAIL response). During the contention period, each node will wait for the Slot Time (35.84 μ s) and then get the status of the channel from the PHY (via the PCS). If the channel is busy, then the node has lost contention and must wait for the next priority resolution period (as determined by the VCS) and the value it currently holds for the contention counter should be stored for the next attempt. If the channel is idle, then the node will decrement the contention counter and wait for the next slot time. If the counter reaches zero and the channel is still idle, then the node has won the contention period, and on the next slot time, it can begin to transmit the frame. The timing for the full channel access procedure is shown in Figure 3.15. Again this is split into two sections, with the upper section showing the interaction between nodes during this process, and the lower section the data as seen on the channel.

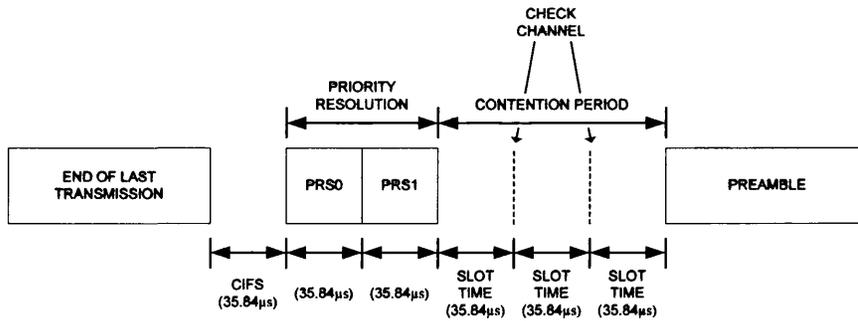
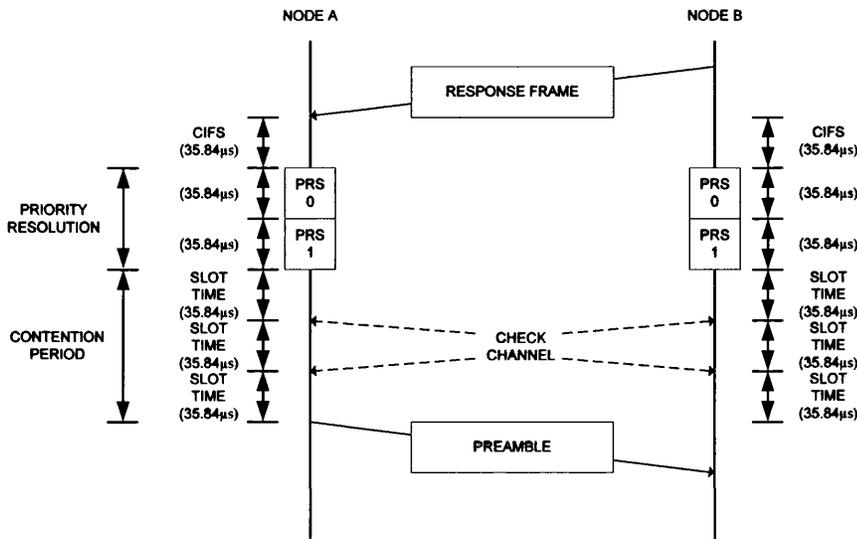


Figure 3.15 - Channel Access Timing

3.3.2.3 MAC Management

The MAC has the ability to send and receive management information, which is done via the MAC Management field of the frame. The field consists of a 2-byte type field, a 1-byte control field and then groups of MAC Management Header, MAC Management length and MAC Management data (collectively know as a MAC Management Entry). The control field contains the number of MAC management entries. Table 3.4 indicates the types of management information that can be transmitted. The last column indicates if the MAC Management can be sent along with the normal data frame.

M-TYPE Value	Interpretation	Prepend to host frame
0 0000	Request Channel Estimation	Allowed
0 0001	Channel Estimation Response	Allowed
0 0010	Vendor Specific	Allowed
0 0011	Replace Bridge Address	Only
0 0100	Set Network Encryption Key	Allowed
0 0101	Multicast with Response	Only
0 0110	Confirm Network Encryption Key	Allowed
0 0111	Request Parameters and Statistics	Allowed
0 1000	Parameters and Statistics Response	Allowed
0_1001 – 0_1111	Reserved on transmit, ignore MME on receive and continue processing service block	No
1_0000 – 1_1111	Manufacturer-specific. Never transmitted on medium	No

Table 3.4 - MAC Management Entries

3.3.3 HOMEPLUG PHY

The HomePlug PHY uses Orthogonal Frequency Division Multiplexing (OFDM) as the modulation scheme, with added error correction blocks for both the Frame Control and Payload. The basic block diagram of the transmitter is shown in Figure 3.16.

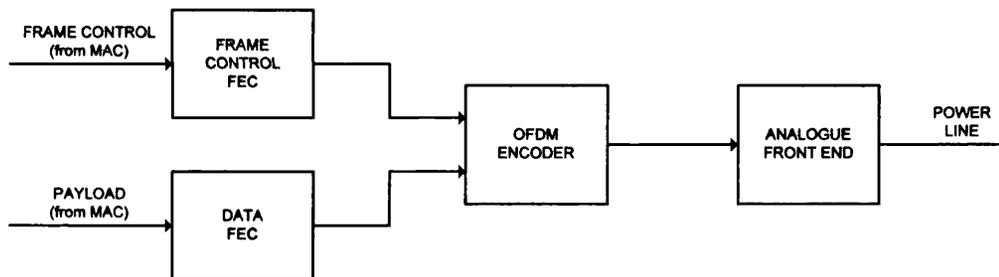


Figure 3.16 - PHY Transmitter Block Diagram

HomePlug has a bandwidth of 25MHz that is split into 128 evenly spaced sub-carriers (i.e. each sub-carrier is 195,312.5 Hz apart), however only those between approximately 4MHz and 20MHz are used to carry data (more exactly frequencies 23 to 106). This gives a maximum of 84 carriers that can be used for data transmission. Not all of these are used however, as some are permanently blocked to allow the system to meet FCC emission regulations [80]. The blocked frequencies are given in Table 3.5, along with the reason that they are blocked. The most common reason is to stop interference with HAM radio bands.

Carrier	Frequency (MHz)	Reason
13	7.03125	40m Amateur Radio Band
14	7.2265625	40m Amateur Radio Band
29	10.15625	Fixed Applications (i.e. public phone lines) Band and 30m Amateur Radio
49	14.0625	20m Amateur Radio Band
50	14.2578125	20m Amateur Radio Band
51	14.453125	Fixed Applications, and close to 20m Amateur Radio
69	17.96875	Aeronautical Mobile/Emergency Band
70	18.1640625	17m Amateur Radio Band

Table 3.5 - Blocked HomePlug Frequencies

Each sub-carrier can be modulated using one of four schemes: Binary Phase Shift Keying (BPSK), Differential Binary PSK (DBPSK), Differential Quadrature PSK (DQPSK) and Robust OFDM (ROBO). In OFDM all sub-carriers must be modulated using the same scheme in any one transmission. There is a version of OFDM called Discrete Multi-Tone or DMT that allows each sub-carrier to use a different modulation scheme in a single transmission, which leads to more complex encoder and decoder circuitry. Of the modulation schemes listed above, BPSK is only used for the frame control and the others only for payload. The choice of modulation scheme gives the number of bits that will be transmitted per sub-carrier, and is given in Table 3.6.

Modulation	Mnemonic	Bits Per Carrier
Binary Phase Shift Keying	BPSK	1
Differential Binary Phase Shift Keying	DBPSK	1
Differential Quadrature Phase Shift Keying	DQPSK	2
Robust OFDM	ROBO	¼

Table 3.6 - Bits Per Carrier for HomePlug Modulations

3.3.3.1 Frame Control FEC

Due to the importance of the Frame Control data (in that it is used by all nodes on the network to determine the network status) it has a very robust FEC, which consists of a Product Encoder and an Interleaver, which are shown Figure 3.17

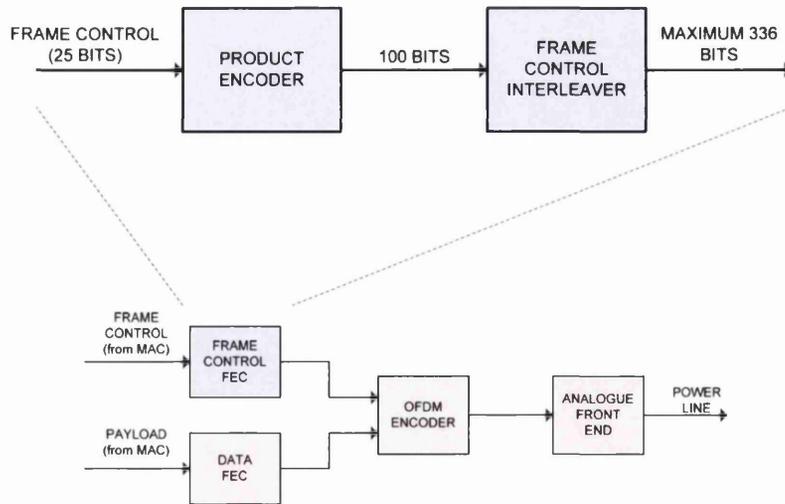


Figure 3.17 - Frame Control FEC Block Diagram

The Product Encoder uses a (100, 25) product code to create 100 encoded bits from the 25 input bits. This is done by placing the input data into a [5x5] matrix and calculating row and column parity using a shortened extended (10,5) Hamming code. This is shown in Figure 3.18, along with the generator matrix that is used.

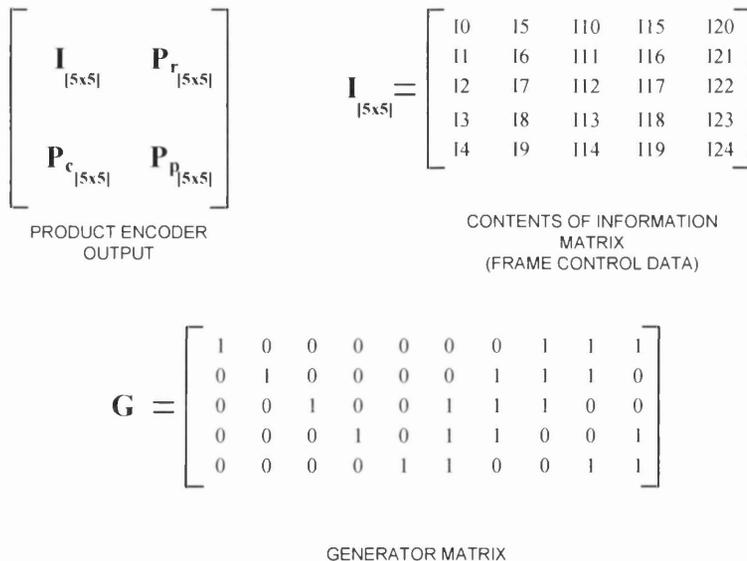


Figure 3.18 - Product Encoder Matrix

The second stage of the FEC is an interleaver, which takes the 100 bits from the product encoder, and interleaves them so that logically adjacent bits aren't transmitted physically adjacent. This improves the systems robustness against burst errors. The interleaver also places the 100 interleaved bits over 4 OFDM symbols. This introduces yet another level

of redundancy as each bit can potentially be transmitted 4 times. The symbols are placed in such a way so that the same bit isn't transmitted on the same carrier in each symbol. Figure 3.19 shows how the interleaved data is spread over the four OFDM symbols.

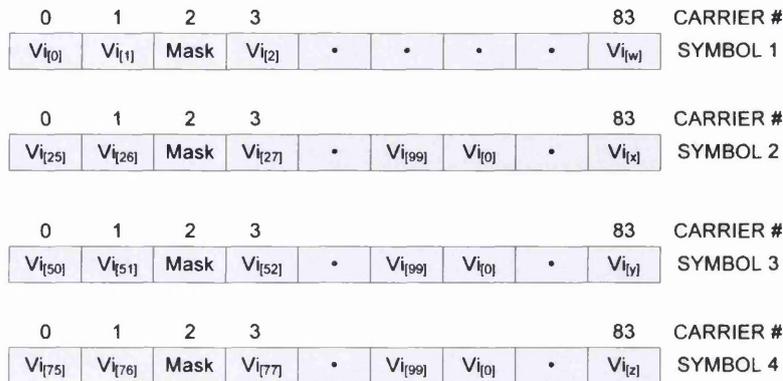


Figure 3.19 - Frame Control Interleaver Bit Spreading

3.3.3.2 Payload FEC

The payload FEC is based around a Reed-Solomon encoder and a Convolutional encoder, along with a bit puncturing block (to give a 3/4-rate convolutional code) and an interleaver. Depending on the modulation technique used by the PHY, one of two interleavers will be used. The block diagram is shown in Figure 3.20.

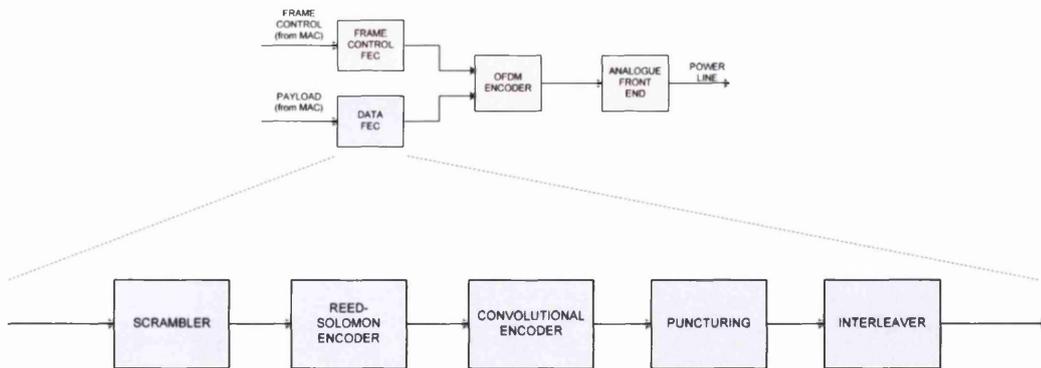


Figure 3.20 - Payload FEC Block Diagram

The first stage of the encoding process is a scrambler. This is used to make the data appear more “random” by exclusive-oring the incoming data stream with a pseudo-random bit stream. This is done to improve the error correcting capability of the RS and convolutional encoder by getting rid of long streams of 0’s and 1’s that might be present in the payload. The block diagram for the scrambler is shown in Figure 3.21.

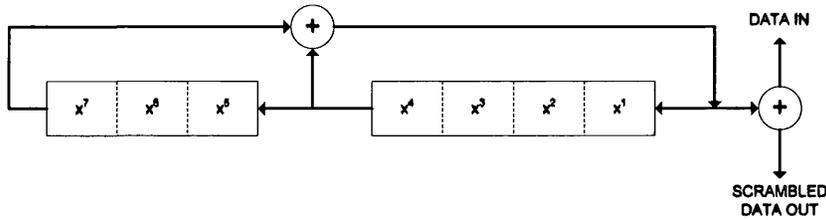


Figure 3.21 - Scrambler Block Diagram

The next stage of the encoding process is the Reed-Solomon (RS) encoder. The RS encoder is a block-based encoder, which uses symbols² of 8-bits to generate parity data that can be used to correct errors. As the code uses 8-bit symbols, the incoming bit stream is grouped into bytes and these are then used in the encoder. They symbols are part of a Galois-field (GF), which has a generator polynomial:

$$f(x) = x^8 + x^4 + x^3 + x^2 + 1.$$

The RS encoder has two modes, one for ROBO modulation and one for DBPSK and DQPSK modulation. They use a different parity code generator, and are capable of correcting a different number of errors. Table 3.7 summaries the key differences and the basic block diagram of the RS encoder is shown in Figure 3.22.

Modulation	Primary RS Code	Parity Symbols	Max. Error Correcting	Min. RS Symbols ³	Max. RS Symbols
DBPSK, DQPSK	RS(255,239)	16	8	23	238
ROBO	RS(255,247)	8	4	31	43

Table 3.7 - Reed-Solomon Modes

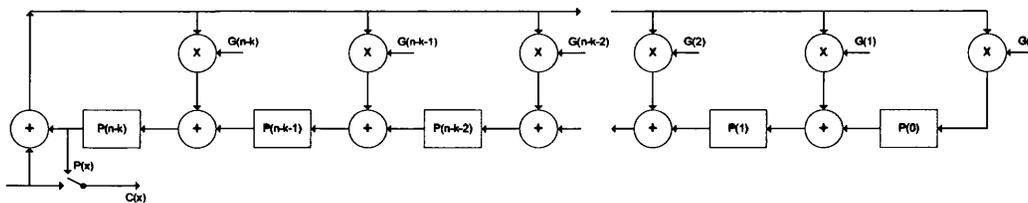


Figure 3.22 - Reed-Solomon Encoder Block Diagram

² These symbols are different to the OFDM symbols introduced previously

³ The RS Encoder used in HomePlug is either a shortened RS(255,238) or RS(255,247) encoder and the Min and Max RS Symbols in the table give the range of RS symbols that will be passed to the encoder (below the full number of symbols the encoder can handle)

The next stage is the Convolutional Encoder, which is a standard $\frac{1}{2}$ -rate, $K=7$ encoder. This means that for every input bit, two bits are output and the value of these depends on the last seven inputs. The block diagram for the encoder is shown in Figure 3.23. At the end of the data stream, the encoder is “flushed” with six zero tail bits, which returns the encoder to its initial state. This improves the performance of the Viterbi decoder.

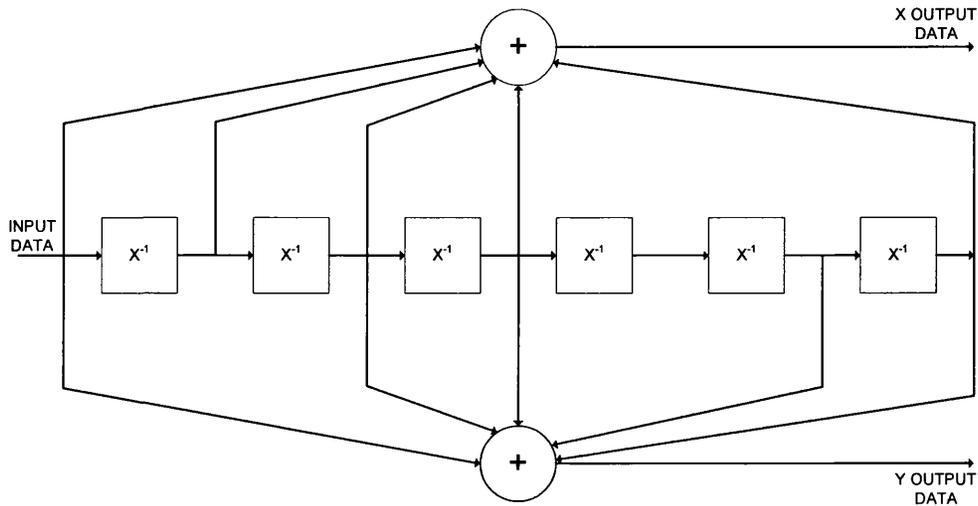


Figure 3.23 - Convolutional Encoder Block Diagram

If the channel conditions are good enough, the $\frac{1}{2}$ -rate data from the convolutional encoder can be “punctured”, which removes bits to reduce the overhead of the encoding thereby allowing more data to be transmitted. This produces a $\frac{3}{4}$ -rate code although the trade-off is a reduced error correcting ability. Figure 3.24 shows the sequence of bit removal that occurs in the puncturer.

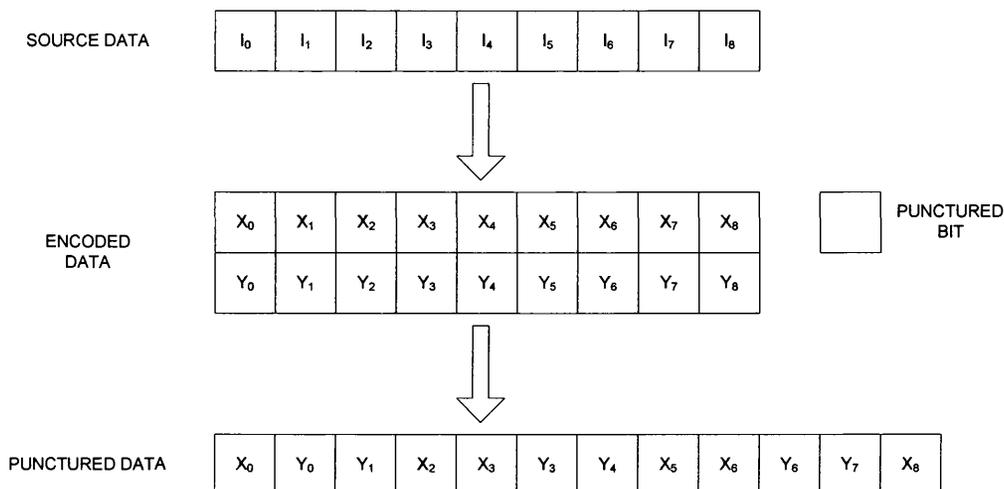
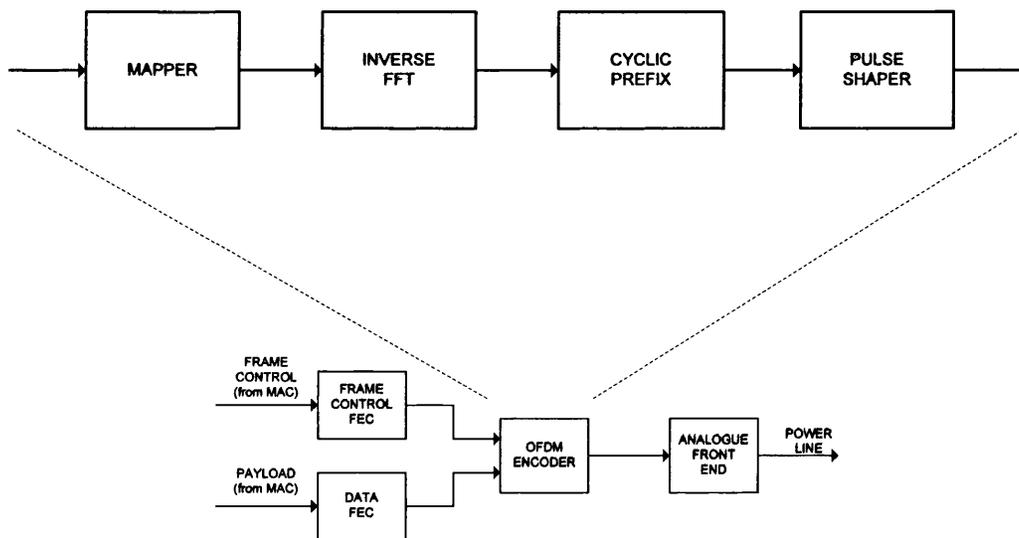


Figure 3.24 - Bit Puncturer

The final stage of the Data FEC is the interleaver. There are two interleaving algorithms, one for ROBO data and one for non-ROBO data. Both are described as row/column block interleavers, as they take the incoming bit stream and place it in a matrix in rows, and read it out in columns. After each column, a shift of 8 is applied. The number of columns is 10 for a 20-Symbol PHY block (or a ROBO block) and 20 for a 40-Symbol PHY block. The number of rows is equal to twice the number of usable carriers. The ROBO interleaver adds additional redundancy by outputting the interleaver matrix four times, and shifting the starting output row between each read.

3.3.3.3 OFDM Encoder

The OFDM encoder consists of a Mapper, an Inverse FFT, a Cyclic Extender and a Pulse Shaper. Figure 3.25 shows the structure of the encoder.

**Figure 3.25 - OFDM Encoder Block Diagram**

The first stage of the OFDM encoding process is the Mapper. This takes the bit stream from either the Frame Control FEC (for the frame control data) or the Payload FEC (for the payload data) and maps it onto the constellations that are used for the transmission. It also ensures that those frequencies that cannot be used either through regulation (i.e. those in the Tone Mask) or from the channel estimation (i.e. those in the Tone Map) do not carry data. In the case of the carriers that are blocked in the Tone Mask, the mapper

will insert a phase of zero and a magnitude of zero (ensuring no power is transmitted on those frequencies). For the carriers that are blocked in the Tone Map, the mapper will insert a pseudo-random binary value, which will be mapped in the same way as the normal data. Only DBPSK and DQPSK modulations obey the Tone Map. The other function of the mapper is to add a reference phase. In the case of BPSK this is the same phase for every sub-carrier. For the other modulations, the reference phase is the phase of the previous symbol at that frequency. The first symbol will use the last Frame Control symbol's phase as its reference. The output of the Mapper is the input data arranged into OFDM symbols, ready for encoding through the IFFT.

The IFFT takes the mapper output (which is arranged as symbols) and places it into the correct frequency "bins" to ensure that the data is transmitted at the correct frequency. In HomePlug this is frequency bins 24 to 106 (or frequencies 4.47MHz to 20.7MHz). It will then perform a 256-point IFFT on this data, which will give an output of 256 samples per symbol.

The 256-samples out of the IFFT are then extended to 428 samples, by taking the last 172 samples of each symbol and placing them at the beginning of the symbol. This is done to reduce the effect of inter-symbol-interference (ISI) as by choosing a cyclic extension that is longer than the longest delay in the channel, it ensures that any interference from the previous symbol is minimised.

The final stage is pulse shaping, which takes the 428 samples from the cyclic prefix block, and applies a Raised-Cosine shape to the pulses. The samples are then sent to the analogue front end before being transmitted on the power-line.

3.4 CHANNEL MODEL

In order to verify the operation of the networking system under realistic conditions, a model of the channel is required. This has two main components, a transfer function and a noise model. The transfer function describes how the signal is attenuated as it travels down the wire. The noise model describes the extra interference that is introduced to distort the transmitted signal between source and destination. These are described in the next two sections. Figure 3.26 shows the basic channel model [81, 82, 83, 84].

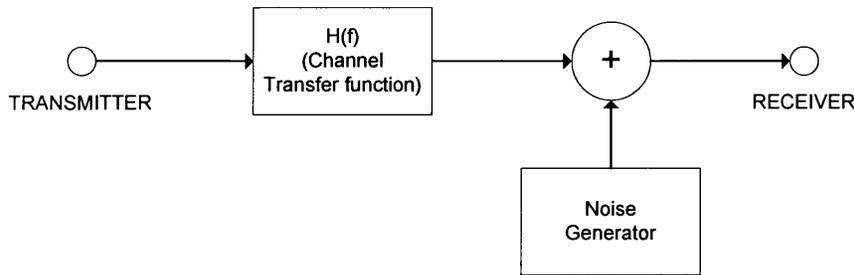


Figure 3.26 – Channel Model

3.4.1 TRANSFER FUNCTION

This can range from a simple point-to-point single path model, to a more complex multi-tap, multi-path model. When modelling a wire or cable, the general method used is a two-port model, which is shown in Figure 3.27 [85].

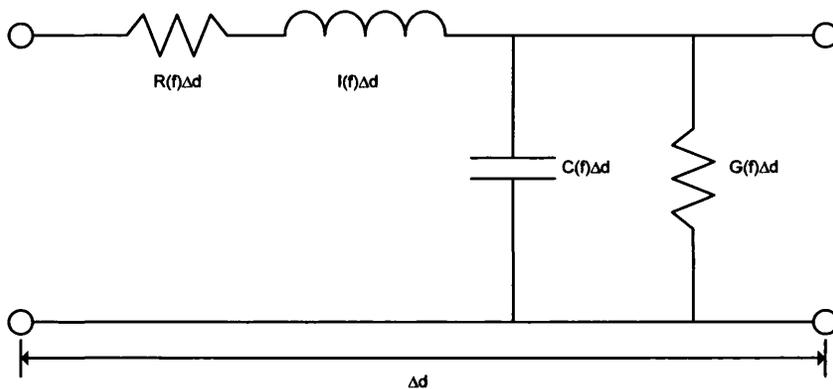


Figure 3.27 - Two-port Model

This describes the length of cable in terms of its characteristic Impedance ($R(f)$), Inductance ($I(f)$), Capacitance ($C(f)$), and Reactance ($G(f)$). These are dependant on the frequency of the signal being transmitted. They also depend on the length of the cable (Δd). A set of equations describing these parameters in terms of their physical properties is given below [85].

$$R(f) = \frac{1}{\pi a} \sqrt{\frac{\pi f \mu_c}{\sigma_c}} \Omega / m \quad L(f) = \frac{\mu}{\pi} \cosh^{-1} \left(\frac{D}{2a} \right) H / m$$

$$G(f) = \frac{\pi \sigma}{\cosh^{-1} \left(\frac{D}{2a} \right)} S / m \quad C(f) = \frac{\pi \epsilon}{\cosh^{-1} \left(\frac{D}{2a} \right)} F / m$$

Where μ_c = Permeability of conductor

σ_c = Conductivity of conductor

a = Radius of conductor

D = Distance between conductors

μ = Permeability of dielectric

σ = Conductivity of dielectric

ϵ = Permittivity of dielectric

From the above equations two important parameters that describe the cable can be determined, the propagation constant, γ , and the characteristic impedance, Z_0 .

$$\gamma = \sqrt{(R + j\omega L)(G + j\omega C)} = \alpha + j\beta$$

$$Z_0 = \sqrt{\frac{(R + j\omega L)}{(G + j\omega C)}}$$

From these the attenuation of the cable can be calculated [86]

$$A(f, d) = e^{-\alpha(f)d} \text{ where } \alpha = \text{Re}\{\gamma(f)\}$$

From this the multi-path transfer function of a network can be found

$$H(f) = \sum_{i=1}^N g_i A(f, d_i) e^{-j2\pi f \tau_i}$$

Where

N is the number of paths to calculate

g_i is a weighting factor for the path, dependant on the nodes its passes through

$A(f, d_i)$ is the attenuation factor for the path

τ_i is the delay of the path and can be determined using $\frac{d_i \sqrt{\epsilon_r}}{c_0}$, where ϵ_r is the permittivity of dielectric and c_0 is the speed of light.

This gives the transfer function for a length of wire. Using a simple network, as shown Figure 3.28, the transfer function from A to C can be calculated [83, 84, 85]

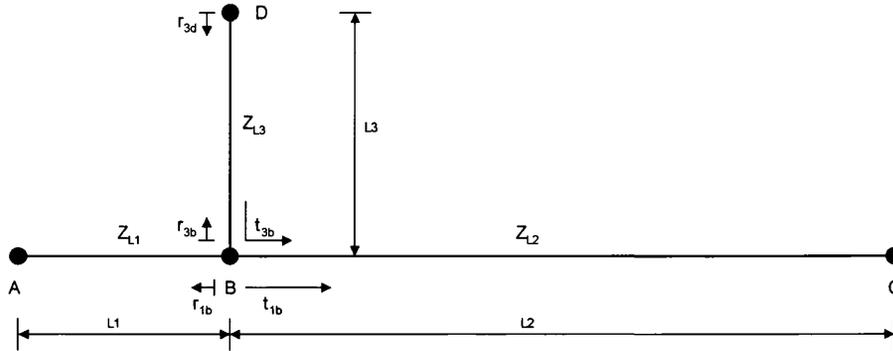


Figure 3.28 - Simple Network Model

Assuming A and C are matched, $Z_A = Z_{L1}$, $Z_C = Z_{L2}$ and therefore don't cause reflection.

The transmission and reflection coefficients, used to calculate g_i , can be defined as

$$r_{1b} = \left(\frac{(Z_{L2} \parallel Z_{L3}) - Z_{L1}}{(Z_{L2} \parallel Z_{L3}) + Z_{L1}} \right) \quad t_{1B} = 1 - |r_{1b}|$$

$$r_{3b} = \left(\frac{(Z_{L2} \parallel Z_{L1}) - Z_{L3}}{(Z_{L2} \parallel Z_{L1}) + Z_{L3}} \right) \quad t_{3B} = 1 - |r_{3b}|$$

$$r_{3d} = \left(\frac{Z_D - Z_{L3}}{Z_D + Z_{L3}} \right)$$

[NB the || indicates a voltage divider operation]

The paths through the network (and their lengths) can be defined as

No, i	Path	Weighting Factor, g_i	Length, d_i
1	A → B → C	t_{1b}	$l_1 + l_2$
2	A → B → D → B → C	$t_{1b} \cdot r_{3d} \cdot t_{3b}$	$l_1 + 2 \cdot l_3 + l_2$
...
N	A → B (→ D → B) ^N → C	$t_{1b} \cdot r_{3d} \cdot (r_{3b} \cdot r_{3d})^{(N-2)} \cdot t_{3b}$	$l_1 + 2(N-1) \cdot l_3 + l_2$

Table 3.8 - Multi-paths Through Network Model

Once these have been calculated, the transfer function can be determined for the link between A and C, which can then be used to determine the impulse response of the channel, which gives the filter-taps required to model the channel as an FIR filter. This method can be extended to create more complicated network layouts.

3.4.2 NOISE MODEL

There are multiple sources of noise in the power-line, and they exhibit different characteristics, however they are all additive (i.e. they will add together to give the final noise figure as seen at the receiver side). Figure 3.29 shows this in relation to the basic channel model (Figure 3.26) [87].

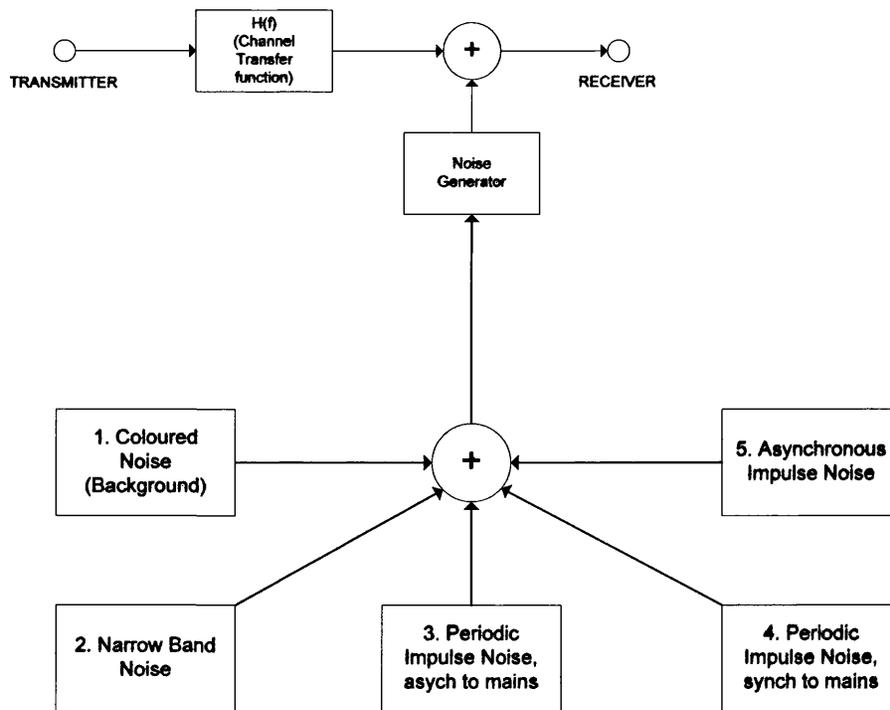


Figure 3.29 – Noise Model

The 5 types of noise are summarised in Table 3.9.

Number	Type	Description
1	Coloured Noise	This is general background noise, caused by many different low-power noise sources, and has a very low Power Spectral Density (PSD).
2	Narrow Band Noise	Sinusoidal signals, mainly caused by broadcasted radio signals.

Number	Type	Description
3	Periodic Impulse Noise, asynchronous to mains	Periodic impulse noises, caused by switching power supplies, with a repetition rate between 50kHz and 200kHz.
4	Periodic Impulse Noise, synchronous to mains	Periodic impulse noises, with a repetition rate equal to the mains frequency. Generally short duration (a few microseconds) and are caused by power supplies operating synchronously with the mains cycle (e.g. dimmer switches)
5	Asynchronous Impulse Noise	Random switching transients in the network, have duration of some microseconds up to a few milliseconds. These can have quite large noise values, sometimes up to 50dB above the background noise.

Table 3.9 – Noise Characteristics [86]

Noise types 3-5 can be thought of as a noise source, which itself is affected by its own channel, so in the case of the power-line, something like an induction motor will introduce noise, however this will be attenuated in the same way as the data signal as it travels from the source to the receiver. This will be a different transfer function to that used by the data (unless the noise comes from the same location as the data signal). The extended version of the noise model is shown in Figure 3.30 [87].

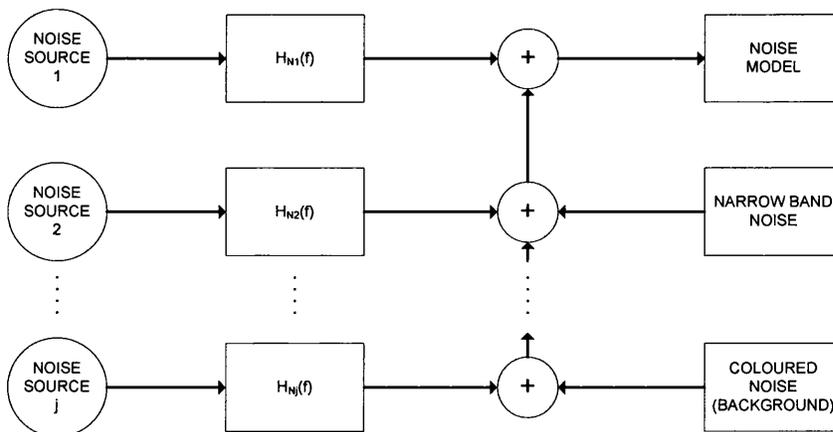


Figure 3.30 – Extended Noise Model

Various methods can be used to describe the noise sources, and for the periodic a common method is Markov models. However these results need empirical evidence to back them up (i.e. experiments carried out to provide the data). Many different people have carried out these experiments; however, they have not supplied enough data to recreate a model of the noise. However the background noise which is fairly simple can be modelled using the following equation

$$N(f) = 10^{(K-3.95e10^{-5}f)} \left(\frac{W}{Hz} \right)$$

This allows a very simplified noise model to be added to the transfer function described above, to create a complete channel function.

3.5 SUMMARY

In this chapter, the necessary background required to understand the rest of the work has been presented. It started with a summary of networking in general, showing the hierarchical, layered approach that is the way in which networks have operated since they were introduced. This allows a focus to be made on a specific area, and also allows any physical network to communicate with a different physical network, for example an Ethernet network can communicate with a wireless network.

The HomePlug powerline standard was then introduced, and the operation of the Media Access Layer (MAC) and Physical Layer (PHY) was described. The HomePlug MAC uses a CSMA/CA scheme for channel access, with an added priority scheme to allow for higher priority traffic. The PHY uses OFDM to encode the data, which makes it robust in the powerline environment.

The chapter closed with a look at a possible channel model, based on an attenuation function and injected noise models. The figures and equations presented are based on others work and no experiments were carried out to back these up.

Chapter 4 - Modelling Environment Requirements and Control

Describes the model that was developed. It first gives the requirements of the model (based on the discussions of previous chapters), and then describes the structure of the model, and how it is controlled. It also introduces the terminology used to describe aspects of the model.

4.1 INTRODUCTION

This chapter builds upon the discussion of previous chapters and introduces the solution that was developed to solve the problem of modelling home networks. The requirements of such a system (based on the discussions of Chapter 2) are given, followed by the top-level model structure and the details of how the model operates. The data processing elements of the model (i.e. those parts that change the binary into properly formatted HomePlug frames) are described in Chapter 5, with this chapter concentrating on the control of the execution of the data processing.

The requirements of the system are given in Section 4.2, the main requirements being the model is a simulation-based, event-driven system. This premise is used to develop the top-level modelling structure that is described in Section 4.3, and the event system described in Section 4.4. Section 4.4 also describes how the event system controls the simulation of the model, along with the description of the System Controller given in Section 4.5. These two things control the Node-Threads (a definition of which is given in Section 4.3) which in turn run the data processing algorithms that implement the HomePlug protocol.

4.2 REQUIREMENTS

From the discussion in Chapter 2, it can be seen that there are many approaches to solving this problem. Therefore before describing the solution that has been developed, it is necessary to state the problem that is being solved as well as the requirements of the solution.

The thesis outlines a development tool to explore System-on-Chip (SoC) based home networking solutions which brings together network modelling and hardware modelling. The development system assists the designers in two ways:

1. How does the networking hardware (i.e. the components on the SoC) interact, and what are the issues of changing the algorithms.
2. How do the nodes on the network interact, as the traffic patterns are different to those found on traditional (office-based) networks, as there will be a greater amount of streaming media.

From the discussions in previous chapters a basic set of requirements was developed. These were split into two areas, mandatory requirements and non-mandatory (i.e. “nice-to-have”) requirements. The mandatory requirements are given in Table 4.1, and the non-mandatory requirements in Table 4.2.

Ease of Use	As the model will be used by non-programmers, it needs to be easily understood so that they can easily explore alternative solutions. This is more important for the protocol specific aspects of the design rather than the general message handling parts, however if new protocols are to be developed then this part will also need to be easily understood.
Event Based	By using events to manage timing of the model, the control will be more realistic [69], and a full sequence of events will not need to be pre-calculated at the start of the simulation.
Simulation Based	By simulating the algorithms rather than using an

	<p>analytical approach, the results will more closely match those from a real system [49]. This also makes it easier to explore alternative algorithms and reduces the time needed to convert the model to actual hardware.</p>
Multiple Nodes	<p>The sort of networks that are being explored are likely to have many nodes attached (say up to 20) and therefore the model has to take into account the effect this will have.</p>
Multiple Levels	<p>As there are up to seven different levels to a networking system, the solution needs to model these (to a greater or lesser degree), and the effect they will have on the data that is being transmitted. For this model, the mandatory networking layers that need to be modelled are the Physical and Data-Link (or Media Access if the IEEE naming convention is used), along with how these interact with the rest of the node (i.e. the interaction with the other network protocol layers running on the SoC).</p>
Data Metrics	<p>In order to provide useful feedback on the design decisions taken in the various components of the network (such as alternate algorithms or different implementations of the protocol) data transfer metrics are needed. These will give information on throughput of the system, frame latency, etc.</p>

Table 4.1 – Modelling System Mandatory Requirements

Multiple Protocols	<p>As the home network solution is unlikely to rely on a single protocol, it is important that the model developed can allow a multi-protocol environment to be explored. The solution given here focuses on a HomePlug based system.</p>
Usable with Hardware Simulators	<p>If the model can be integrated within the actual hardware simulation environment then this will allow it to be used as a reference model to aid verification, or</p>

allow design engineers to use it to generate data as input to the part they are developing (for example using the MAC model to create accurate frames as input to a PHY design).

Table 4.2 – Modelling System Non-Mandatory Requirements

Given these requirements, there is the important choice of implementation language. An initial Physical layer model was developed in Matlab; however this was slow and didn't allow the modelling of the interaction between multiple nodes within the system. Also it would have been difficult to use this within hardware simulators. There are many possible languages that could have been chosen, such as C, C++, Java, System C, BONEs, NS2, etc., however the model was developed using C with the pthreads library [90]. The pthreads library was used to provide the event handling/concurrent features required. The decision was taken to implement the model in C as this was the most readily available language and the one the developer was most familiar with. It is also highly portable, as all that is needed is a C compiler and the pthread library. A Linux based PC with GCC was used as the development machine.

4.3 TOP LEVEL MODEL STRUCTURE

The model consists of a System Controller, multiple nodes and a channel. Figure 4.1 shows these components and how they interact.

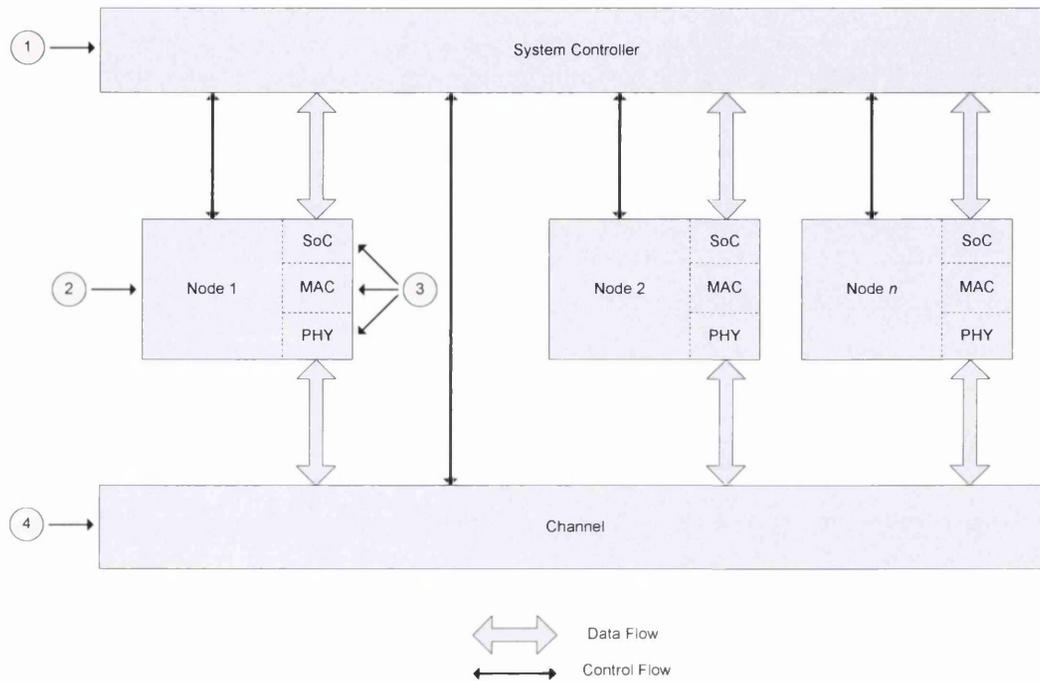


Figure 4.1 – Model Structure

To ease confusion, a set of terminology to describe the various parts of the model is introduced. Table 4.3 gives the terms used to describe elements within the model, and the numbers highlight these in relation to Figure 4.1.

Element	Number	Description
System Controller	1	Controlling thread of the model.
Node	2	A “Device” on the network, which consists of a SoC, MAC and PHY
Node-Thread	3	Sub-component of the node. Independent thread that performs the tasks of the node. It’s one of Soc, MAC or PHY.
Channel	4	Transmission medium, consisting of a single Transmit channel-thread and multiple Receive channel-threads.

Table 4.3 – Model Terminology

The System Controller, the node-threads and the channel-threads all run as independent threads. This allows them to operate in “parallel” (although there will only be one thread

running at any single time on a single CPU computer). The reason for this is to allow the node-threads to be performing different tasks simultaneously. For example, the PHY node-thread could be decoding a frame, whilst the MAC node-thread is assembling the next one for transmission.

There are two aspects to the model; the protocol specific part and the system interaction part. The protocol part is responsible for encoding and decoding the messages that are being sent through the system according to the details of the protocol (see Section 4.3 for details of this for the HomePlug protocol). The system interaction part is responsible for ensuring the various events/actions happen at the correct “time”, so that the model will reflect a real-life system.

The System Controller is responsible for ensuring events occur at the correct time. It also starts and stops the nodes (or more correctly the node’s node-threads). The System Controller is actually the “main” function (or more exactly is a while loop within the “main” function), and one of its tasks is to read and parse the command file at start-up. This file gives the instructions to be carried out for the test/simulation that is being run. The format of each line in this file, which is parsed when the is run, is

<INSTRUCTION> <TIME> <OPTIONS>

The instructions, and a description, are given in Table 4.4. The options depend on the instruction, and these are also given in Table 4.4.

Instruction	Description	Options
CMD_STOP	Stops the simulation run. Is always the last command in the file	None
CMD_ADDNODE	Adds a new node to the model, and starts it	<Node ID Number> <RX Buffers> <RX Buffer Priority> <TX Buffers>
CMD_DELNODE	Stops/deletes an existing node from the model	<Node ID Number>
CMD_TX	Initiates the transmission of a message from one node to another	<Source Node> <Destination Node> <Priority> <Length>
CMD_TX_STREAM	Initiates the transmission of a data stream	<Source Node> <Destination Node> <Priority> <Length> <Data Rate>
CMD_NOCHAN	Indicates that no channel characteristics are to be used	None
CMD_USECHAN	Use a specified set of channel characteristics	<Channel Characteristic Directory>

Table 4.4 – Model Command File Instructions

The CMD_TX_STREAM Command is used to model data sources (traffic models). The command file parsing function uses the data about the length and data rate and converts it into a number of “TX” events, sufficient to transmit the data at the rate specified.

A full description of how the System Controller operates is given in Section 4.5.1.

A potential use of the modelling system is to allow engineers to explore various alternative implementations/algorithms, and compare their effectiveness (using whatever criteria is relevant, i.e. speed, area of the hardware, power consumption, accuracy, etc.). This could potentially have a big impact on the final hardware, and it is important to explore the alternatives before the design progresses to the hardware implementation stage. An example of this is given in Chapter 6, where the multipliers used in the PHY are replaced with logarithmic multipliers. These sacrifice accuracy for area and speed improvements [91]. A further example is given using different buffer sizes and seeing the effect this has on latency.

4.4 MESSAGE/EVENT SYSTEM

4.4.1 EVENT SYSTEM OVERVIEW

To allow the various node-threads within the model to communicate with each other (where they are permitted to) an event handling mechanism was developed which allows events to be passed between the node-threads and between the controller and the node-threads. The events that can be handled along with the sequence of events are described in the first part of this section. The second part details the mechanisms that are used to send the events.

Figure 4.2 shows the flow of events between the node-threads, and between the node-threads and the controllers. Nodes cannot communicate events directly to each other, as all transmissions of this sort have to follow the HomePlug rules. In this case the “events” are passed as data through the channel, and the only valid event is “NEW_DATA”. The events that are sent between components are given in the next section.

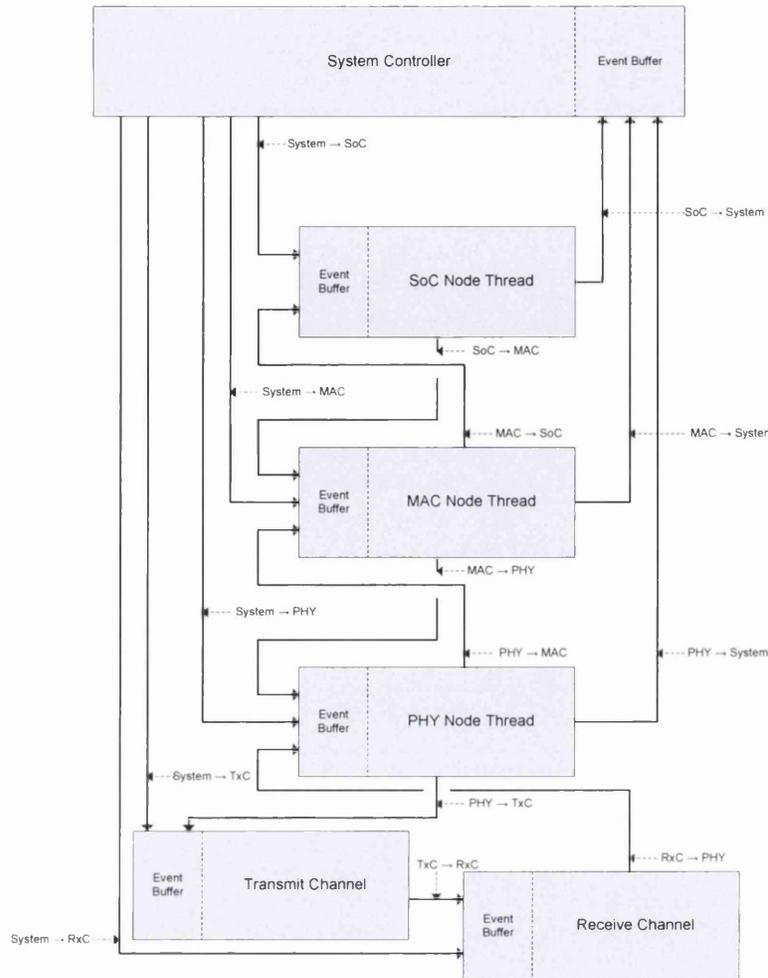


Figure 4.2 – Event Communications

The features of the event system are listed below:

- Each node-thread has its own time, controlled by the incoming events.
- Each node-thread holds a list of expected events, based on past events, to ensure that “unknown” events are processed at the correct time.
- If an unknown (unexpected) event occurs, then the node-thread will decide if the event is processed or stored.
- Most events have associated future events, for example, instructing the MAC to assemble a frame results in a channel access event being expected by the controller.
- Most events cause the node-thread to process some data, which will take an associated amount of time, and this will be reflected in the next event.

4.4.2 SYSTEM EVENTS AND SEQUENCES

A description of all the events used within the system is given in Table 4.5. The actions that are taken when these occur are given in Sections 4.5 and 4.6. The table is grouped into the events that occur between the various components within the model, as shown in Figure 4.2.

Event	Description
<i>System to SoC Events</i>	
STOP	Stops the SoC Node Thread
START	Starts and initialises the SoC Node Thread
NEW_FRAME	Requests the SoC to begin the message/frame assembly process by telling the MAC to begin frame assembly
<i>System to MAC Events</i>	
STOP	Stops the MAC Node Thread
START	Starts and initialises the MAC Node Thread
SLOT_TIME	Tells the MAC to check the status of the channel during the random back off process
PRS	Tells the MAC to initiate the Priority resolution process
<i>System to PHY Events</i>	
STOP	Stops the PHY Node Thread
START	Starts and initialises the PHY Node Thread
TRANSMIT	Tells the PHY to transmit the next OFDM symbol on the channel
<i>System to Transmit/Receive Channel</i>	
STOP	Stops the transmit/receive channel
START	Starts and initialises the transmit/receive channel
LOAD	Tells the channel to load the filter coefficients from the filter data structure
<i>SoC to MAC Events</i>	
AHB	Indicates a transfer over the AHB Bus
APB	Indicates a transfer over the APB Bus
<i>MAC to SoC Events</i>	
AHB	Indicates a transfer over the AHB Bus
INT	Indicates an interrupt from the MAC to the SoC (for a new message reception)
APB	Indicates a transfer over the APB Bus
<i>MAC to PHY Events</i>	
PRS	Tells the PHY to transmit a PRS symbol over the channel
CHAN_STATE	Tells the PHY to check the status of the channel
NEW_FRAME	Tells the PHY that there is a frame ready for encoding
<i>PHY to MAC Events</i>	
PRS	Result of the PRS symbol – indicates to the MAC if the node has won or lost that symbol
CHAN_STATE	Tells the MAC what the status of the channel is
GOT_FRAME	Tells the MAC that the PHY has a decoded frame
<i>PHY to Transmit Channel Events</i>	
NEW_DATA	Tells the Transmit Channel that there is a new symbol ready for filtering and transmission
<i>Transmit to Receive Channel Events</i>	
NEW_DATA	Tells the Receive Channel that there is a new symbol ready for filtering and transmission to the PHY
<i>Receive Channel to PHY Events</i>	
NEW_DATA	Tells the PHY that a new symbol has been received

Event	Description
<i>PHY to System Events</i>	
READY	Tells the System Controller that the PHY has encoded the frame and is ready to transmit it at the correct time
LAST_DATA	Tells the System Controller that the next symbol is the last for this frame
RXD	Tells the System Controller that the PHY has received the current OFDM symbol
<i>MAC to System Controller Events</i>	
SLOT_TIME	Tells the System Controller what the outcome of the random back off was for the MAC concerned
ADD_BO	Tells the System Controller that the node didn't lose priority resolution and so needs to begin the random back off
GOT_FRAME	Tells the System Controller the MAC has a frame, and needs to transmit a response
ADD_PRS	Tells the System Controller that the MAC has a frame to transmit and needs a PRS event at the time specified
<i>SoC to System Controller Events</i>	
GOT_FRAME	Tells the System Controller that the SoC has received a frame

Table 4.5 – Modelling System Events

These events allow the model to initiate and control transmission of frames/data between nodes. There are five phases needed to achieve this, and each has its own sequence of events, which are described here. The five phases are:

1. Transfer and encoding of the data to the MAC
2. The channel access procedure
3. Transfer of the frame to the PHY, encoding and transmission over the channel
4. Decoding of the frame and generation of the response frame
5. Transferring the data to the receiving SoC

The transfer of the data to the MAC (phase 1) is started by the Controller sending a NEW_FRAME event to the SoC, which then gets the data to be transmitted, and sends it to the MAC via the APB/AHB. When the MAC has assembled the PHY frames, it will send an ADD_PRS event to the Controller so that at the correct time, the MAC can initiate the channel access procedure. Figure 4.3 shows this sequence.

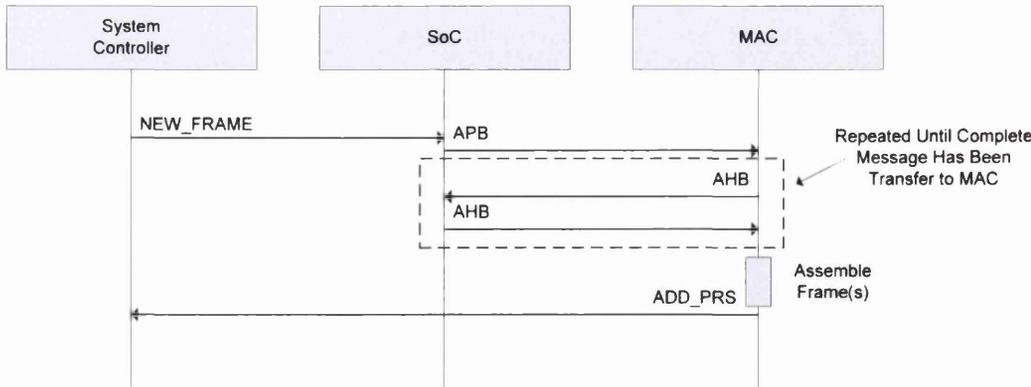


Figure 4.3 – Phase 1 Event Sequence

The channel access procedure (phase 2) follows the rules of the HomePlug standard (given in Section 3.3.2.2). At the correct time (the next priority resolution period), the MAC will begin priority resolution. If the MAC doesn't lose this, it will then begin the random back-off process. At the end of this, if the node has won, it will send a "WON_BO" event to the Controller, so that it can transmit the frame at the correct time. The sequence of events for phase 2 is shown in Figure 4.4.

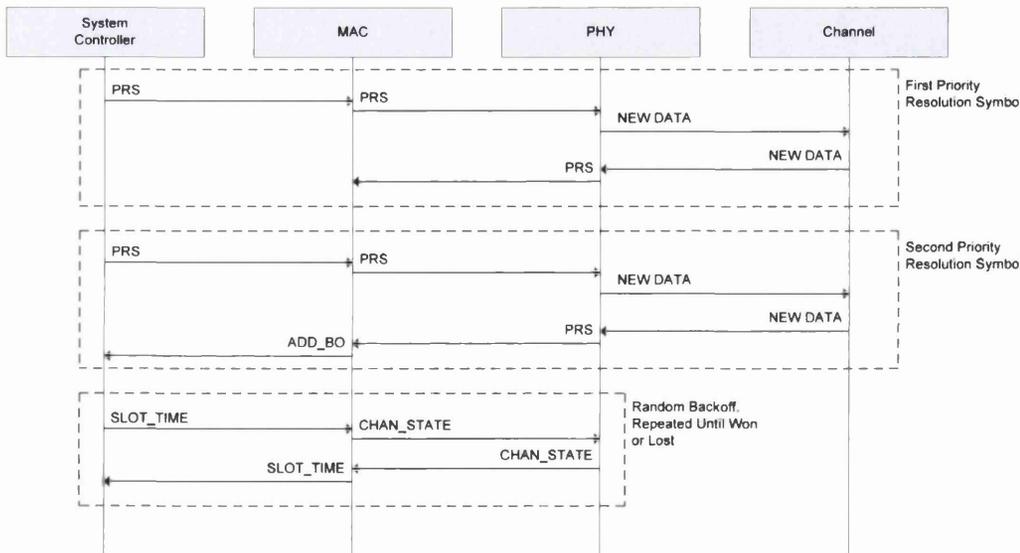


Figure 4.4 – Phase 2 Event Sequence

The next stage of the process is for the MAC to transfer the frame to the PHY, which encodes it and then transmits it over the channel to the other nodes in the system. The transmission is done OFDM symbol by OFDM symbol. This allows the controller to

change the characteristics of the channel during transmission if needed. The sequence of events for phase 3 is shown in Figure 4.5.

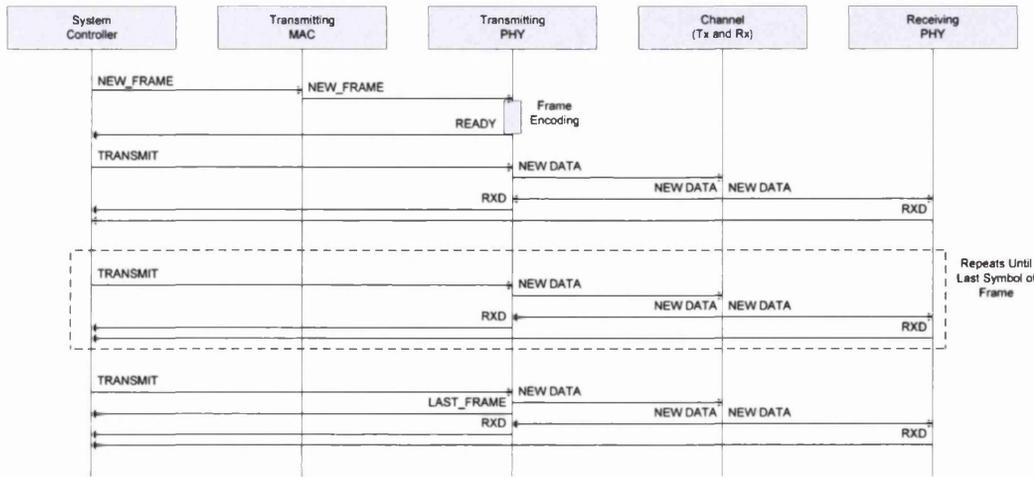


Figure 4.5 – Phase 3 Event Sequence

Stage four involves the receiving node decoding the frame, and generating the appropriate response. Once the response is generated, the receiving PHY will signal to the Controller, which will then signal the PHY to transmit the response. The sequence of events for phase 4 is shown in Figure 4.6. Before checking the frame, the receiving MAC will also check if it has sufficient buffer space for the node. If it doesn't then it will generate a FAIL response.

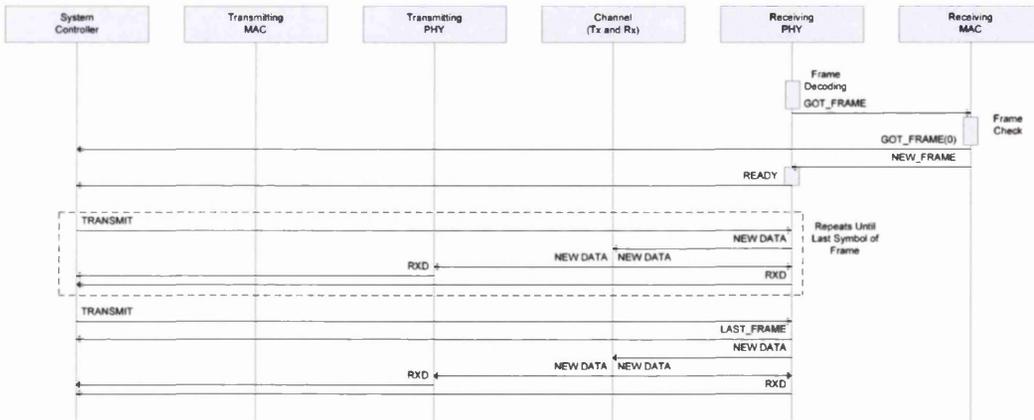


Figure 4.6 – Phase 4 Event Sequence

The final phase is transferring the received message from the receiving MAC to the receiving SoC, once all the frames have been received (this is given by the Last Segment flag in the Segment Control field of the received frame). This is the inverse of the transmission phase, and involves the MAC generating an interrupt, and then transferring

the data over the APB/AHB back to the SoC which then checks that the data is correct. The sequence of events is shown in Figure 4.7.

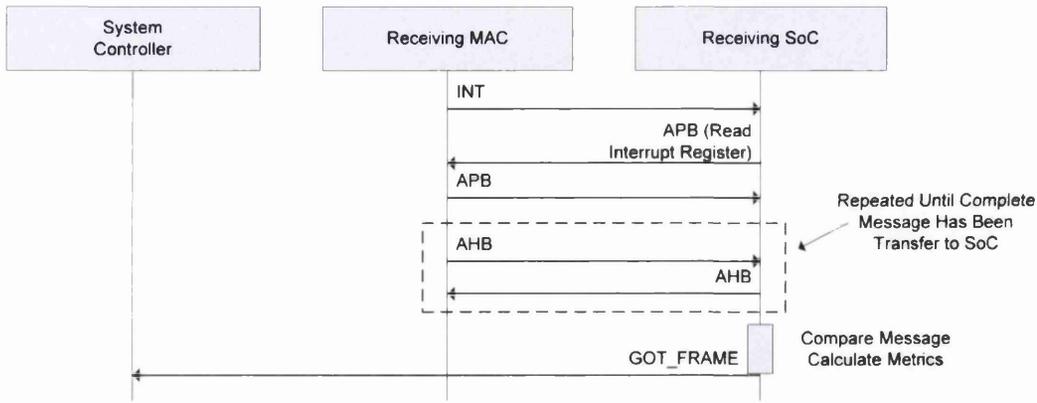


Figure 4.7 – Phase 5 Event Sequence

4.4.3 EVENT STRUCTURE

The events system described above is implemented using a thread-safe data structure, and a set of functions to send and receive data. There are three main functions, and two auxiliary functions associated with the event structure. The main functions are:

1. Initialisation – Clears the data in the event data structure, and creates the mutual exclusion lock and semaphore.
2. Send an Event – Stores the event information and then signals the receiving thread (via the semaphore) that there is an event waiting.
3. Wait on an Event – Blocks the thread and waits for the send event to signal that an event has occurred. It then copies the event data back to the calling thread.

The auxiliary functions are for debug purposes and print the event that has occurred to the screen or to a file.

The event structure follows a “Producer-Consumer” model [90], where a single thread will use the “Wait on an Event” function (the consumer) and other threads will use the “Send an Event” function (the producers). This allows multiple threads to communicate with each thread, for example the MAC thread can respond to events sent from the controller thread, the SoC thread and the PHY thread. The event structure consists of the control devices needed, plus the actual data. The event structure is given in Table 4.6, and the event data is given in Table 4.7. The functions themselves use a pointer to the event structure in order to manipulate the data within.

Field	Type	Description
lock	pthread_mutex_t	Event structure mutual exclusion lock
sem	pthread_sem_t	Event semaphore, indicates a new event
data	event_data	The event data
cond	pthread_cond_t	Conditional variable for the empty flag
empty	int	Flag to indicate the event data is empty

Table 4.6 – Event Structure

Field	Type	Description
id	int	The ID of the node sending the event
time	int	The “time” of the event
desc	event_desc	The description of the event
dir	event_dir	The direction of the event. Internal (i.e. MAC→PHY) or External (i.e. Controller→PHY)
src	int	The source node (for transmitting data)
dest	int	The destination node (for transmitting data)
msg_id	int	The message to send (for transmitting data)
flag	int	Variant field, its meaning depends on the event
fname	char[]	The filename

Table 4.7 – Event Data Structure

The algorithm for the initialisation function (`event_init`) is as follows.

1. Create the mutual exclusion lock (protects the event and data structures)
2. Create the conditional variable (allows threads to block waiting on the event structure becoming empty)
3. Create the semaphore
4. Clear each element of the event data structure
5. Set the “Empty” flag (allows threads to send events)

The parameters to the function are:

```

event_t *eventp      A pointer to the actual event structure
event_attr_t *attrp  Event structure attributes. These are not
                    used in this version of the structure

```

The algorithm for the send function (`event_send`) is as follows:

1. Lock the mutual exclusion lock. This prevents other threads using structure
2. Wait until the data structure is empty
 - 2.1. If it is not, release the mutex lock, and block until it is
3. Copy the input data to the event data structure
4. Send the semaphore indicating that an event is waiting
5. Unlock the mutual exclusion lock. This allows other threads to now use the structure

The parameters to the function are:

```

event_t *eventp      A pointer to the actual event structure
event_data data_in   The event data to send

```

The algorithm of the wait function (`event_wait`) is as follows:

1. Block waiting for the semaphore
2. Lock the mutual exclusion lock
3. Copy the event data to the function output
4. Set the "Data Empty" flag, and send the indication so waiting threads can send their event
5. Release the mutual exclusion lock

The parameters to the function are:

<code>event_t *eventp</code>	A pointer to the actual event structure
<code>event_data *data_out</code>	The received event data

4.4.4 EVENT HANDLING WITHIN NODE-THREADS

Using the event structure described above, the various threads can communicate with each other and pass information about events that are occurring. As the time an event happens could be some time in the future (from the time the node-thread is at) each node-thread holds a list of expected and pending events. This means that if an event arrives with a time after the next pending or expected event, then the node-thread will store the event. This situation could occur depending on how the node-threads are scheduled, which depends on the underlying operating system.

Within each node-thread the basic algorithm is as follows

1. While the STOP event hasn't occurred
 - 1.1. Determine what event to process or block waiting for an event
 - 1.2. Set the time of the node-thread to the time of the event (if one is being processed)
 - 1.3. Process the event

Step 1.1 is common across all node-threads and will be described here. Step 1.3. depends on the node-thread (i.e. Soc, MAC, and PHY) and the specifics will be described in subsequent sections.

As mentioned above, each node-thread uses the same algorithm to determine what event to process, based on the status of two lists:

1. Pending Events List – Those events that have arrived at the node-thread, but haven't been processed.
2. Expected Events List – The events that haven't arrived at the node-thread, but are expected to some time in the future.

The algorithm used to determine what event to process is given below.

1. If there are no pending events or the next pending event is after the next expected event
 - 1.1. Wait on the event (`event_wait`)
 - 1.2. Determine if the arriving event will be processed or pended. If there are no expected events, or the event is the next expected event or it occurs before the next expected event
 - 1.2.1. Set the process event flag

- 1.2.2. If the event is the expected on, remove it from the expected event list
(event_llist_get)
- 1.3. Else the event should be pended
 - 1.3.1. Add the event to the pending list (event_llist_insert)
- 2. Else process the next pending event
 - 2.1. Get the event from the pending list (event_llist_get)
 - 2.2. Set the process event flag
 - 2.3. If the pending event is the next expected event
 - 2.3.1. Remove it from the expected event list (event_llist_get)

The event lists are based on a singly-linked list, which is sorted on the time of the event. When an event is added to a list the function searches for the correct place in the list to insert the event, before inserting the event into the list and updating the list pointers. When an event is removed from the list, the event at the start of the list is always returned (unless the list is empty). There are also various auxiliary functions used to compare events, etc.

The functions for managing the linked lists are given in Table 4.8, and the algorithms for the primary functions (initialise, add and remove) are given.

Function	Description
event_llist_init	Initialises the linked list
event_llist_insert	Inserts the event into the correct place in the list
event_llist_get	Gets the event at the start of the list
is_empty	Checks if the list is empty
is_same	Checks if the event passed in is the same as the one at the start of the list
is_same_sys	Checks if the event passed in is the same as the one at the start of the list. Version for the system controller, as “same” means different things depending on the event
is_same_list	Checks if the events at the start of the two lists passed in are the same
is_after	Checks if the start of list 1 is after (in time) the start of list 2
is_before	Checks if the event passed in is before (in time) the start of the list

Table 4.8 – Event Linked List Functions

The parameters to the initialisation function are:

event_llist *elp The pointer to the event list

The algorithm for the initialisation function is

- 1. Make the first node in the list NULL

2. Return success

The parameters to the Insertion function are:

<code>event_llist *elp</code>	The pointer to the event list
<code>event_data data</code>	The data to insert into the list

The algorithm for the insertion function is

1. Get the first node in the list
2. If the list is empty (first node is NULL)
 - 2.1. Create space for the new node
 - 2.2. Copy each element of the event structure to the node
 - 2.3. Set the first node of the list to point to the new node
3. Else if the new event occurs before the first node
 - 3.1. Create space for the new node
 - 3.2. Copy each element of the event structure to the new node
 - 3.3. Make the “next” node of the new node the current first node of the list
 - 3.4. Make the first node of the list the new node
4. Else search through the list to find the correct location
 - 4.1. While the new node position has not been found, and the end of the list hasn’t been reached
 - 4.1.1. If the next node is after the event
 - 4.1.1.1. Set the found flag
 - 4.1.2. Else
 - 4.1.2.1. Move to the next node in the list
 - 4.2. Create space for the new node
 - 4.3. Copy each element of the event structure to the new node
 - 4.4. If a location in the list was found
 - 4.4.1. Set the new node’s “next” node to the current node’s “next” node
 - 4.5. Else
 - 4.5.1. Set the new node’s “next” node to NULL
 - 4.6. Set the current node’s “next” node to the new node

The parameters to the Get event function are:

<code>event_llist *elp</code>	The pointer to the event list
<code>event_data *data</code>	The data at the start of the list

The algorithm for the get event function is:

1. If the list is empty

- 1.1. Return failure
2. Else
 - 2.1. Get the first node in the list
 - 2.2. Copy each element of the event structure to the new node
 - 2.3. Update the first node in the list to be the next node
 - 2.4. Free the space used by the old first node
 - 2.5. Return success

The rest of the functions are simple ones that do the tasks given in the description in the table. When checking if two events are the same, the function checks that the `time`, `id`, `desc` and `dir` fields are the same.

These functions and the algorithms given above for the event handling give the mechanism in which events are managed within the system. The next section describes the actions each node-thread takes when it processes one of the events given in Section 4.4.2.

4.5 SYSTEM CONTROLLER

4.5.1 OVERVIEW

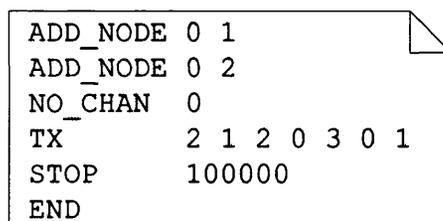
The system controller is more complex than the node-threads, as it also has to initiate events at the correct time. Consequently it doesn't use the same event handling system that the node-threads use, but a modified one to allow it to also initiate new events (such as starting the transmission of a frame).

The main functions of the system controller are:

- Creating space in memory for the event system data structures, data transfer structures and the node-threads.
- Parsing the command file at start-up.
- Starting and stopping the nodes.
- Initiating message transfers.

4.5.2 COMMAND FILE PARSING

Once the system controller has initialised its event structures, it parses the command file to determine the steps it should take during the simulation. As mentioned in Section 4.3 the command file is a list of command and options associated with them. An example command file is given in Figure 4.8.



```
ADD_NODE 0 1
ADD_NODE 0 2
NO_CHAN 0
TX      2 1 2 0 3 0 1
STOP    100000
END
```

Figure 4.8 – Example Command File

The algorithm used to parse the command file is as follows.

1. Open the file "test.cmd" in the directory given
 - 1.1. Return an error if the open fails
2. Read the first command
3. While the command isn't "END"
 - 3.1. Convert the command string to an enumerated type

- 3.2. Read the time of the command from the file
- 3.3. If the command is "CMD_STOP"
 - 3.3.1. Create the list entry for a STOP event at the specified time
 - 3.3.2. Add the event to the event list (`event_llist_insert`)
- 3.4. If the command is "CMD_ADDNODE"
 - 3.4.1. Read the Node ID from the file
 - 3.4.2. Create the list entry for an ADD_NODE event at the specified time, with the "src" field set to the ID from the file
 - 3.4.3. Add the event to the event list
- 3.5. If the command is "CMD_DELNODE"
 - 3.5.1. Read the Node ID from the file
 - 3.5.2. Create the list entry for a DEL_NODE event at the specified time, with the "src" field set to the ID from the file
 - 3.5.3. Add the event to the event list
- 3.6. If the command is "CMD_TX"
 - 3.6.1. Read the source and destination nodes from the file
 - 3.6.2. Read the message length from the file
 - 3.6.3. Read the priority from the file
 - 3.6.4. Create the list entry for a TX event at the specified time, with the source and destination nodes, message ID and flag (with information from 3.6.3) from the file
 - 3.6.5. Add the event to the event list
- 3.7. If the command is "CMD_TX_STREAM"
 - 3.7.1. Read the source and destination from the file
 - 3.7.2. Read the message length, priority and data rate
 - 3.7.3. Calculate the number of frames needed
 - 3.7.4. Calculate the time between frames
 - 3.7.5. For each frame
 - 3.7.5.1. Add a TX event at the specified time (as for 3.6.4) to the event list
- 3.8. If the command is "CMD_NOCHAN"
 - 3.8.1. Create the list entry for a NO_CHAN event at the specified time
 - 3.8.2. Add the event to the event list (`event_llist_insert`)
- 3.9. If the command is "CMD_USECHAN"
 - 3.9.1. Read the directory containing the channel definitions
 - 3.9.2. Create the list entry for an USE_CHAN event at the specified time
 - 3.9.3. Add the event to the event list (`event_llist_insert`)
- 3.10. Read the next command from the file
4. Close the file
5. Return success

4.5.3 SYSTEM CONTROLLER EVENT SYSTEM

As it is the System Controller that initiates events, it needs a list of the events that are to occur as well as those that are expected from the node-threads. When the model starts this is filled with data from the command file, however as the model is executing certain incoming events will cause new events to be added to the outgoing event list. For example, after an “ADD_BO” event, a “SLOT_TIME” event will be added to the outgoing list.

Although the event system is more complex, as the controller initiates events as well as responding to incoming ones, the controller uses the same event list and event passing structures and mechanisms that the node-threads do. However, when processing events, the outgoing list needs to be checked to determine if the event is one from the outgoing or incoming lists.

The algorithm for determining this is quite complex, and is shown in the flow chart in Figure 4.9. The “Process Outgoing Events” and “Process Incoming Events” steps are described in detail in the next section.

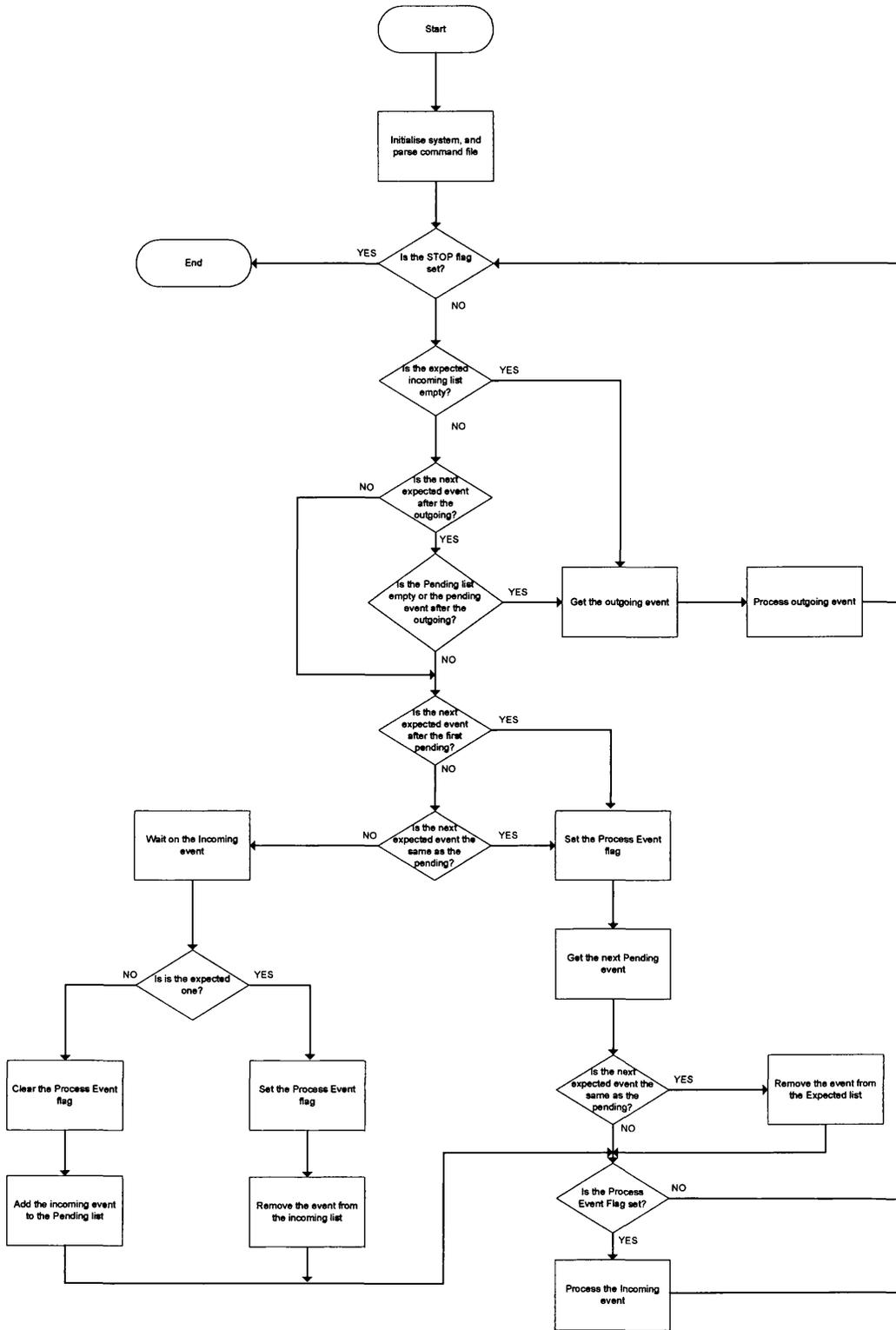


Figure 4.9 – System Controller Operation

4.5.4 SYSTEM CONTROLLER EVENT PROCESSING ALGORITHMS

As mentioned previously, the system controller can process outgoing and incoming events (as shown in Figure 4.9). The steps taken for each of these are shown in the Nassi-Shneiderman diagrams [92] in Figures 4.10 to 4.12.

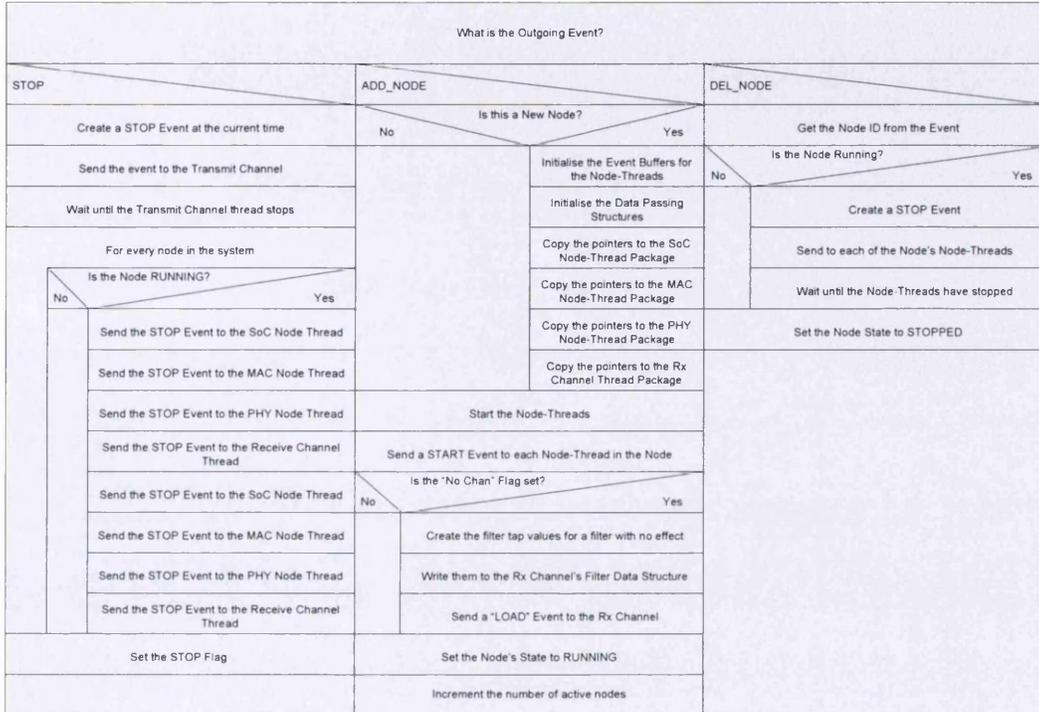


Figure 4.10 – Outgoing Event Nassi-Shneiderman Diagram 1

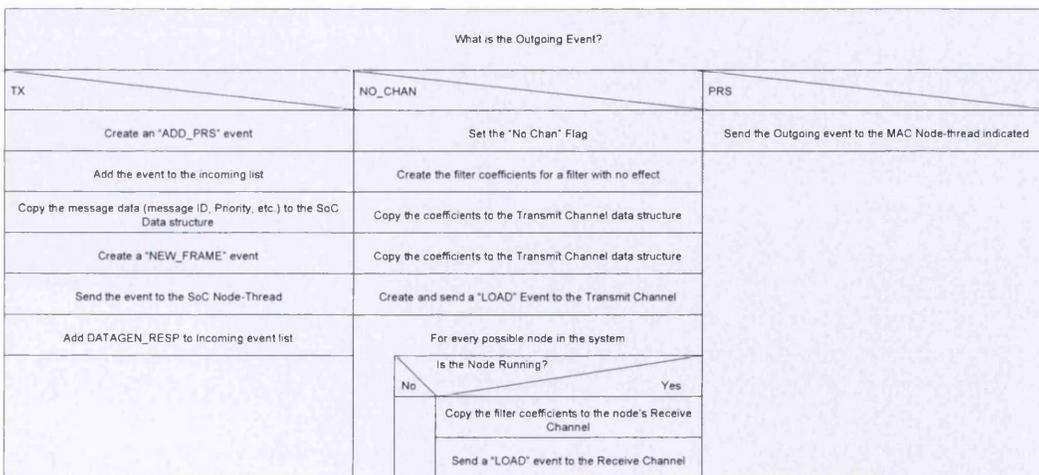


Figure 4.11 – Outgoing Event Nassi-Shneiderman Diagram 2

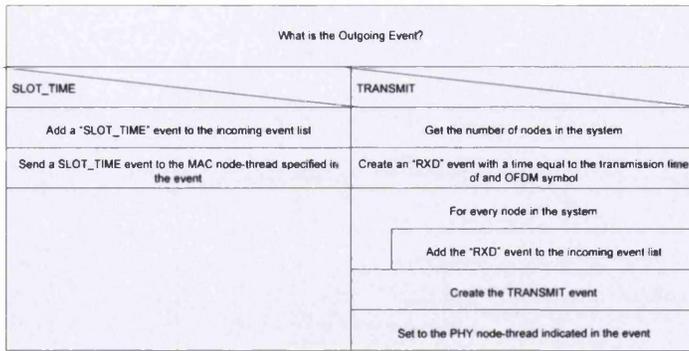


Figure 4.12 – Outgoing Event Nassi-Shneiderman Diagram 3

Figures 4.13 and 4.14 describe the steps the controller takes when processing incoming events.

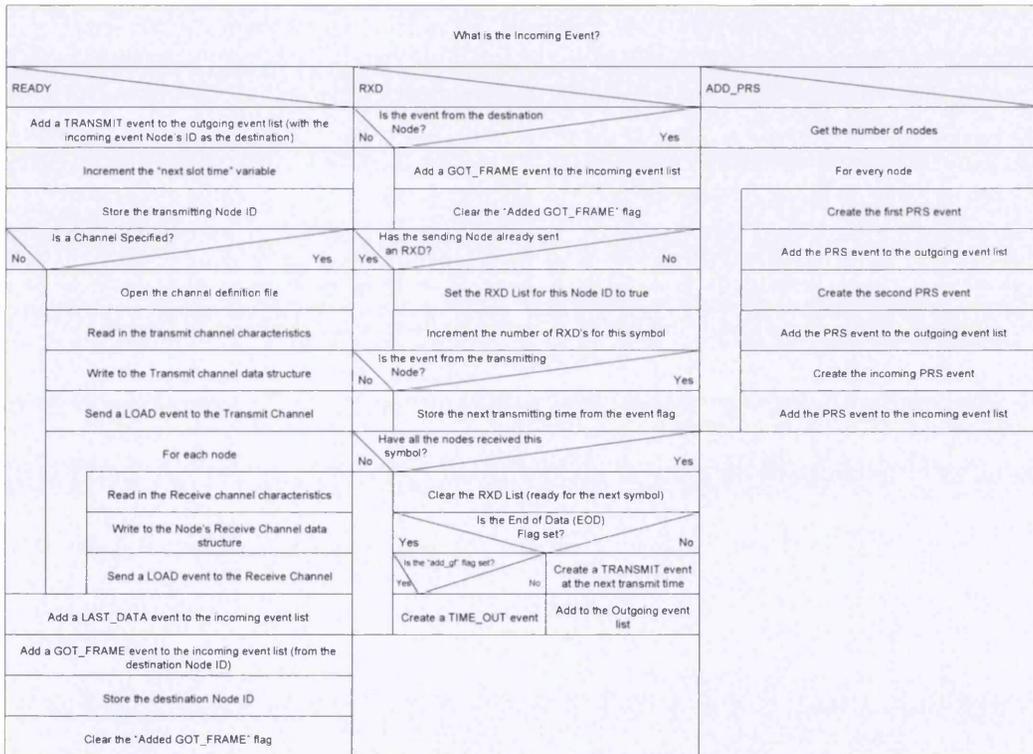


Figure 4.13 – Incoming Event Nassi-Shneiderman Diagram 1

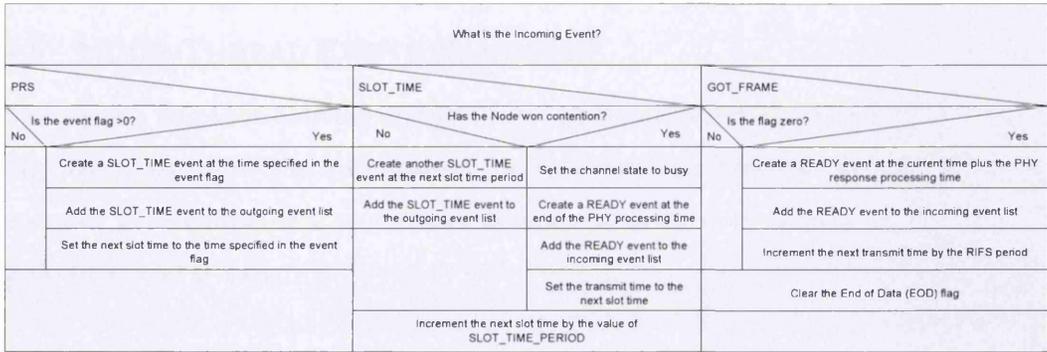


Figure 4.14 – Incoming Event Nassi-Shneiderman Diagram 2

4.6 NODE-THREAD EVENT HANDLING

This section describes how the node-threads handle the events that occur, and the steps they take to process the information. It also describes how the System Controller operates. The controller is much more complex than the node-threads as it also has to generate events as well.

4.6.1 SOC NODE-THREAD

This section describes how the SoC component of the node operates, and how it responds to the incoming events from the System Controller and the MAC, which are given in Table 4.10. The actions that are carried out for each event are described below. When the SoC node-thread is started, various pointers are passed to it to provide it with the information it needs to operate. This is done via the SoC package variable, and the contents of this are given in Table 4.9.

Variable Name	Type	Description
id	int	Node's ID number
event_buf	event_t *	Pointer to the SoC's event buffer
sys_event_buf	event_t *	Pointer to the System Controller's event buffer
mac_event_buf	event_t *	Pointer to the MAC's event buffer
info	sys_info *	Pointer to the System Information structure
info_struct	rdwr_struct_t *	Pointer to the RDWR Information structure
ahb	ahb_t *	Pointer to the AHB model
apb	apb_t *	Pointer to the APB model
soc_data	rdwr_soc_t *	Pointer to the Data structure for the SoC to create the frame to transmit
output_dir	char *	Output directory for log and debug files
log_file	FILE *	File ID for log files
debug_file	FILE *	File ID for debug files

Table 4.9 – SoC Thread Package Structure

Event	Description
STOP	Stops the MAC node-thread
START	Starts the MAC node-thread
NEW_FRAME	Indicates that there is a frame to transmit over the channel, and the SoC needs to send it to the MAC
APB	Indicates a transfer over the APB bus
AHB	Indicates a transfer over the AHB bus
INTERRUPT	Indicates that an interrupt has occurred

Table 4.10 – SoC Events

The algorithm used to process each event is given in the Nassi-Shneiderman diagrams in Figures 4.15 and 4.16.

4.6.2 MAC NODE-THREAD

This section describes how the MAC component of the node operates, and responds to the incoming events from the System Controller, SoC and PHY, given in Table 4.12. The process that is carried out for each event is also given. When the MAC node-thread is started, various pointers are passed to it to provide it with the information it needs to operate. This is done via the MAC package variable, and the contents of this are given in Table 4.11.

Variable Name	Type	Description
id	int	Node's ID number
event_buf	event t *	Pointer to the MAC's event buffer
sys_event_buf	event t *	Pointer to the System Controller's event buffer
soc_event_buf	event t *	Pointer to the SoC's event buffer
phy_event_buf	event t *	Pointer to the PHY's event buffer
info	sys_info *	Pointer to the System Information structure
info_struct	rdwr_struct t *	Pointer to the RDWR Information structure
macphy	rdwr_macphy t *	Pointer to the MAC/PHY Data transfer structure
tm_tran	rdwr_tm t *	Pointer to Tone Map transfer structure
ahb	ahb t *	Pointer to the AHB model
apb	apb t *	Pointer to the APB model
output_dir	char *	Output directory for log and debug files
log_file	FILE *	File ID for log files
debug_file	FILE *	File ID for debug files

Table 4.11 – MAC Thread Package Structure

The HomePlug algorithm for channel access is implemented at this level, as it relies heavily on timing interaction between the various node-threads and the system controller. This is an implementation of the theory given in Section 3.3.2.2.

Event	Description
STOP	Stops the MAC node-thread
START	Starts the MAC node-thread
APB	Indicates a transfer over the APB bus
AHB	Indicates a transfer over the AHB bus
PRS	Priority Resolution response (from the PHY)
SLOT_TIME	Used during the Random Back-off period to tell the MAC to check the status of the channel
CHAN_STATE	Channel status response (from the PHY)
GOT_FRAME	Indicates that the PHY has a fully decoded frame

Table 4.12 – MAC Events

The algorithm used to process each event is given in the Nassi-Shneiderman diagrams in Figures 4.17 to 4.19.

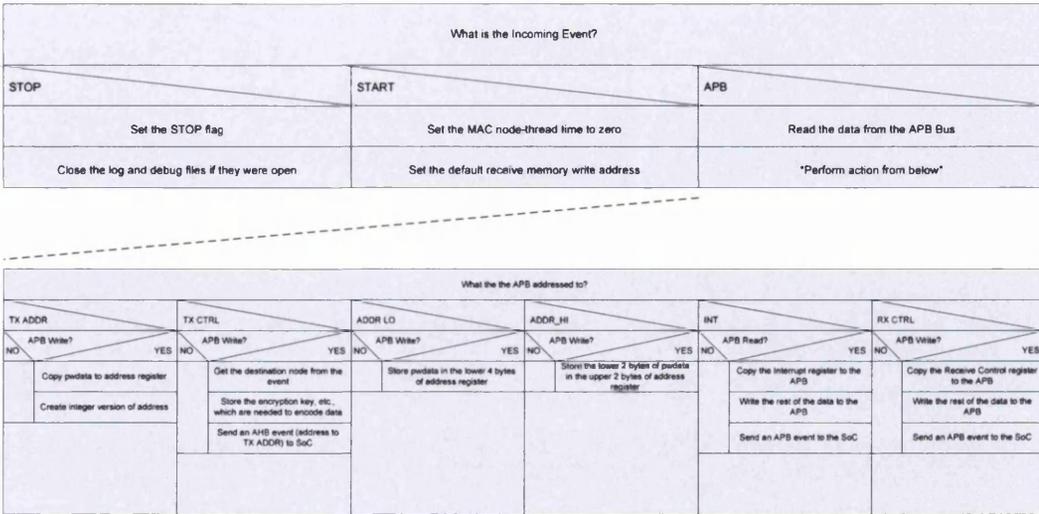


Figure 4.17 – MAC Event Nassi-Shneiderman Diagram 1

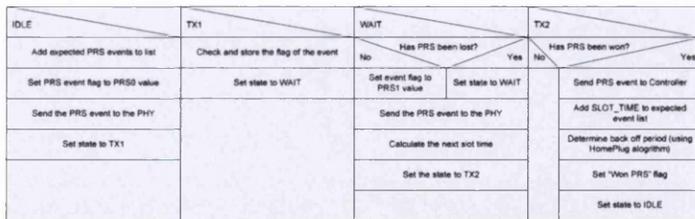
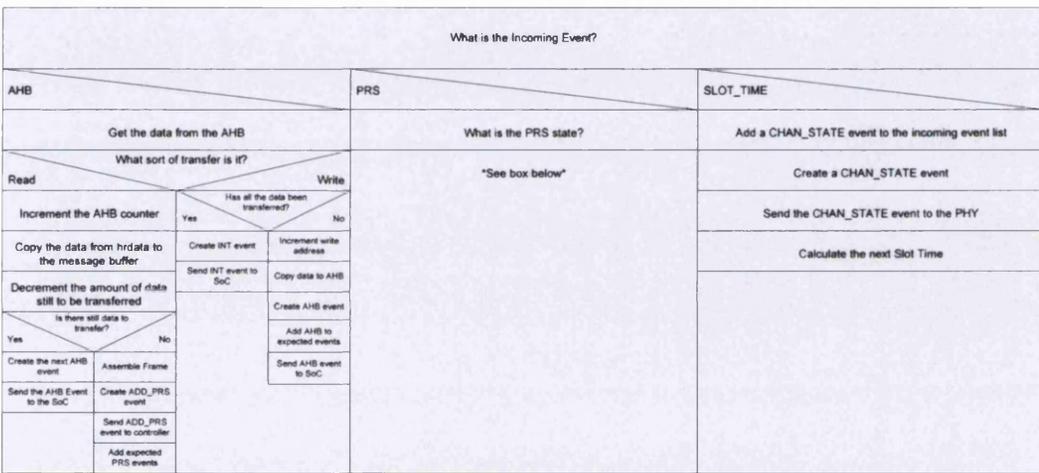


Figure 4.18 – MAC Event Nassi-Shneiderman Diagram 2

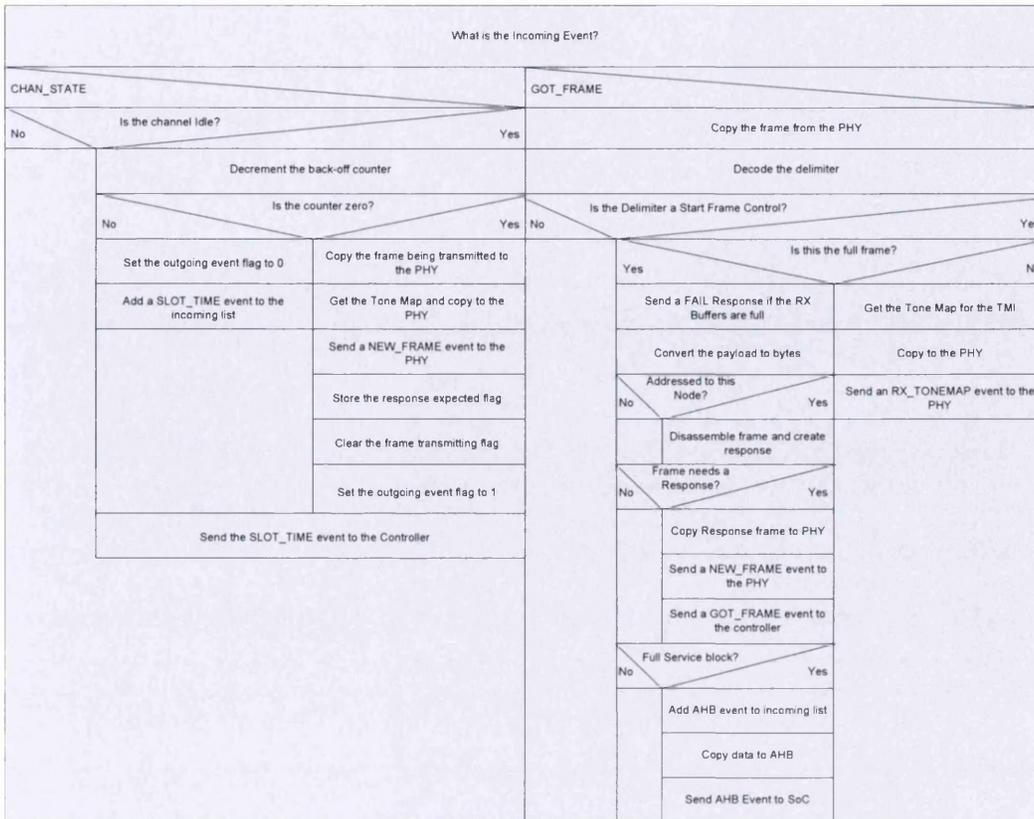


Figure 4.19 – MAC Event Nassi-Shneiderman Diagram 3

4.6.3 PHY NODE-THREAD

This section describes how the PHY operates, and responds to the incoming events given in Table 4.14. The process that is carried out for each event is also given. When the PHY node-thread is started, various things are passed in to it via the PHY Package variable. The details of this are given in Table 4.13.

Variable Name	Type	Description
id	int	Node's ID number
event_buf	event t *	Pointer to the PHY's event buffer
sys_event_buf	event t *	Pointer to the System Controller's event buffer
txc_event_buf	event t *	Pointer to the Transmit Channel's Event buffer
mac_event_buf	event t *	Pointer to the MAC's event buffer
info	sys_info *	Pointer to the System Information structure
macphy	rdwr_macphy t *	Pointer to the MAC/PHY Data transfer structure
txc_data	chan t *	Pointer to PHY to Channel data transfer structure
rx_data	chan t *	Pointer to Channel to PHY data transfer structure
tm_tran	rdwr_tm t *	Pointer to Tone Map transfer structure
output_dir	char *	Output directory for log and debug files
log_file	FILE *	File ID for log files
debug_file	FILE *	File ID for debug files

Table 4.13 – PHY Thread Package Structure

Event	Description
STOP	Stops the node-thread
START	Starts the node-thread
NEW_FRAME	Indicates that there is a PHY frame ready for encoding and transmission
TRANSMIT	Tells the PHY to transmit the next OFDM symbol
NEW_DATA	Tells the PHY that there is an OFDM symbol on the channel
PRS	Request to tell the PHY to transmit a PRS symbol (the value is encoded in the event flag)
CHAN_STATE	Instructs the PHY to check the status of the channel (from the MAC)
RX_TONEMAP	Indicates that the Tone Map for the receiving frame is available

Table 4.14 – PHY Events

The algorithm used to process each event is given in the Nassi-Shneiderman diagrams in Figures 4.20 to 4.22.

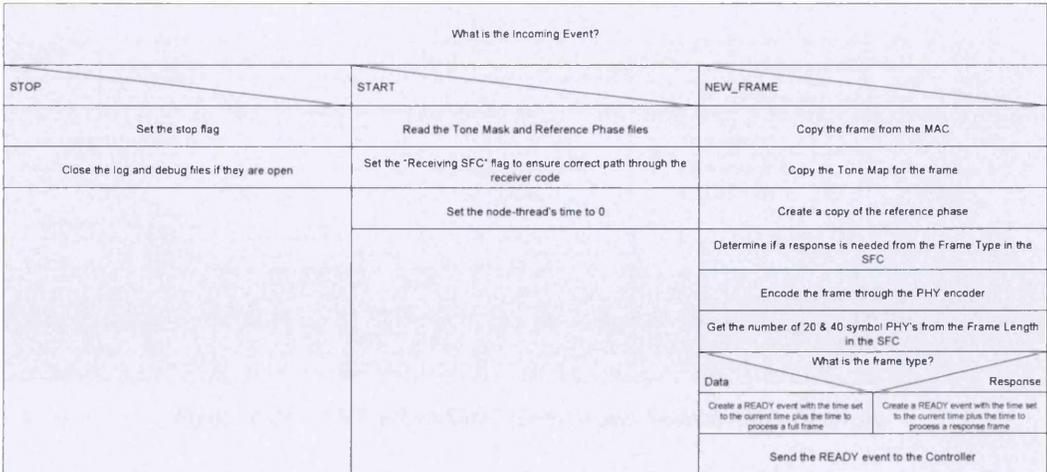


Figure 4.20 – PHY Event Nassi-Shneiderman Diagram 1

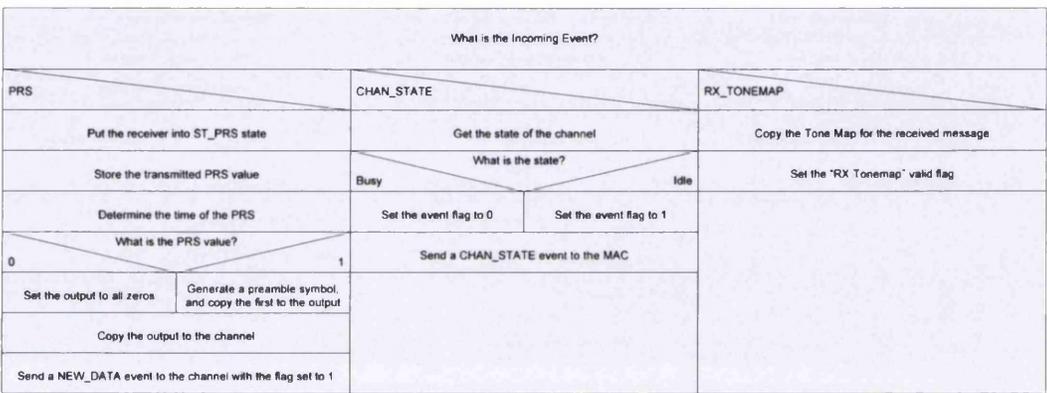


Figure 4.21 – PHY Event Nassi-Shneiderman Diagram 2

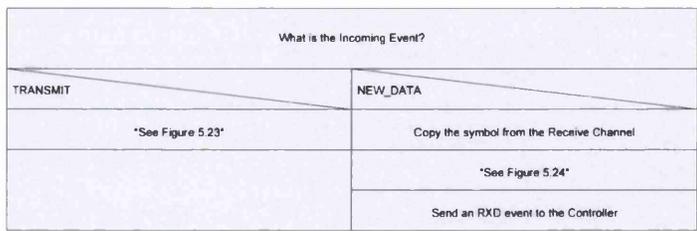


Figure 4.22 – PHY Event Nassi-Shneiderman Diagram 3

As the functionality for the TRANSMIT and NEW_DATA events is complicated, the algorithms are described in Figures 4.23 and 4.24.

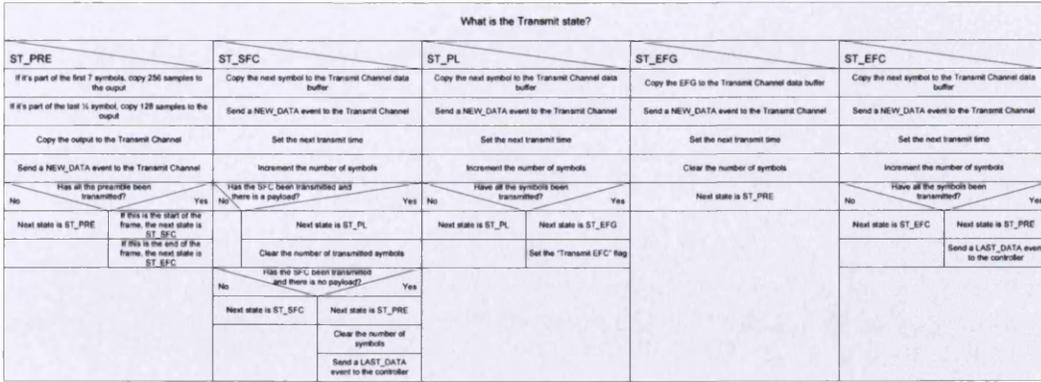


Figure 4.23 – PHY TRANSMIT Event Nassi-Shneiderman Diagram

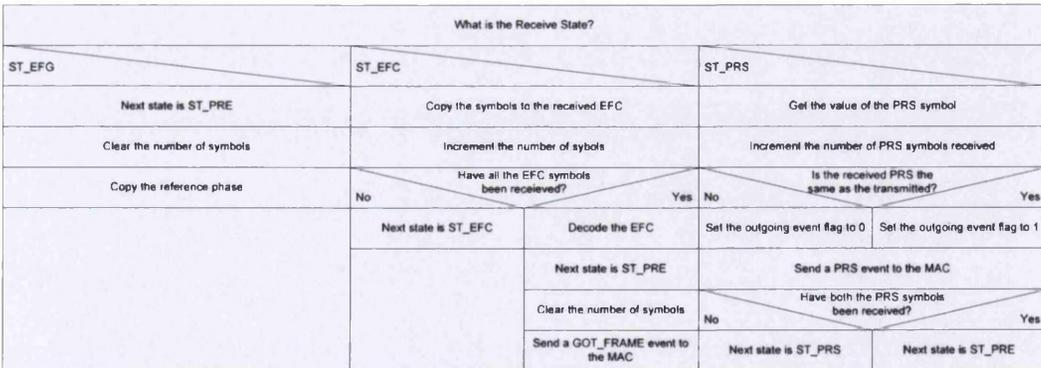
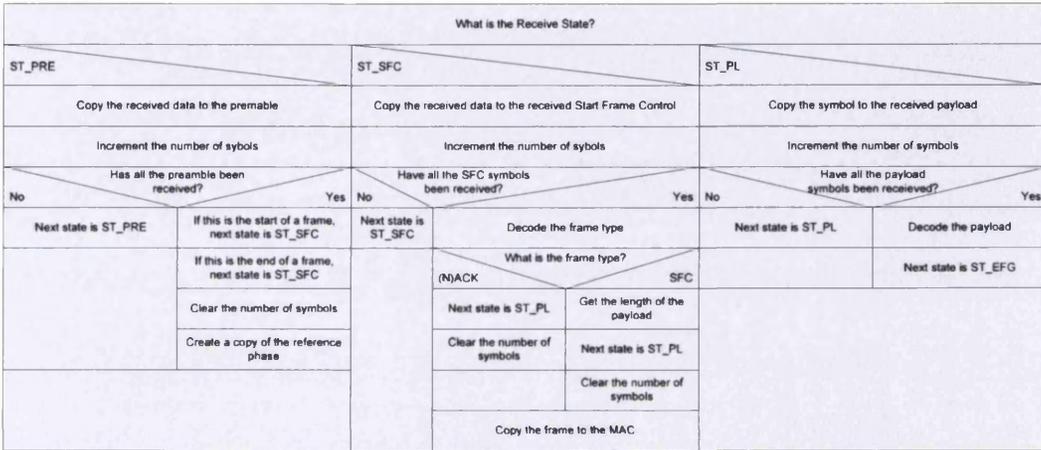


Figure 4.24 – PHY NEW_DATA Event Nassi-Shneiderman Diagram

4.7 SUMMARY

In this chapter, the basic requirements of the modelling system were introduced, namely an event-based simulation model of a home networking system. This was used to describe a top-level structure of the system developed, which consists of a System Controller, multiple Nodes (which have separate Node-Threads to model the SoC, MAC and PHY components of the Node), and a Channel.

The event system used to pass events between the threads within the system was described, along with the events that can be passed. Once the events had been described the actions the components of the model take when receiving these events were described.

The structure described here is the command and control aspects of the model. The data processing parts (i.e. the parts that are more specific to the HomePlug protocol), which build upon these parts, are described in the next chapter.

Chapter 5 - Modelling Environment HomePlug Components

Continues the description of the model, with the focus being on the way the data is modified by the algorithms. It starts with a description of the MAC functions, before describing the PHY and finally the channel.

5.1 INTRODUCTION

In the previous chapter the way the model is controlled and behaves was described. This chapter focuses on how the model takes the data to be transmitted, and encodes it according to the HomePlug standard. It describes an implementation of this standard, although without the Channel Access parts of the MAC.

The chapter is split into three main sections.

1. The Media Access (MAC) functions.
2. The Physical Layer (PHY) functions.
3. The channel functions.

The functions developed follow the specification given in the HomePlug standard, for both transmission (encoding) and reception (decoding). The standard gives detailed descriptions of the encoding functions; however, less information (almost none), is given on the decoding functions. Often the decoding functions are more complicated, especially for the error correcting ones.

The encoding and decoding functions developed can be used independently of the rest of the model, as they don't rely on any of the time/message handling functionality that was described in the previous chapter. This means they could be used to generate a properly formatted PHY frame for example.

5.2 MAC MODEL

Figure 5.1 shows the components that implement the HomePlug specific parts of the MAC. The MAC takes the message to send from the SoC and creates the frames that the PHY encodes before transmission on the channel. At the receiver this process is reversed, and additionally the frame is checked to determine the correct response frame to send.

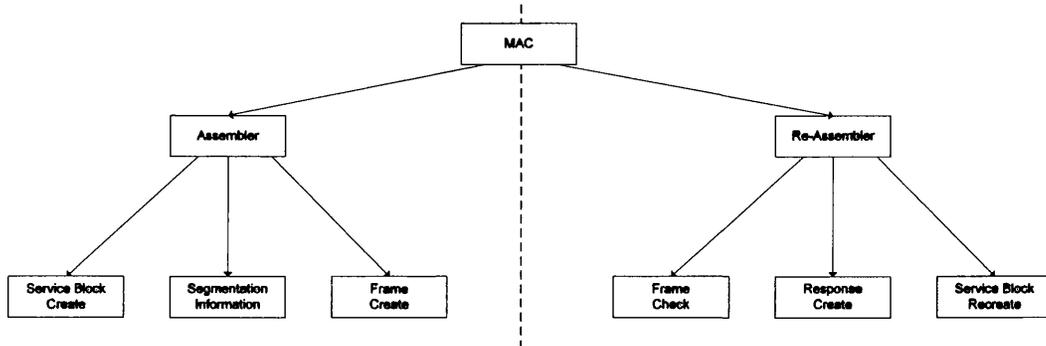


Figure 5.1 – MAC Structure Chart

5.2.1 MAC FRAME ASSEMBLER (ASSEMBLE)

The Frame Assembler takes the data from the SoC and generates the correct frame(s) to send to the PHY. It consists of three sub-functions, which are given below.

1. Service Block Create (`servb_create`)
2. Segment (`segment`)
3. Frame Create (`frame_create`)

The inputs and outputs to the function are given in Table 5.1.

Variable Name	Type	Direction	Description
<code>frames</code>	<code>macphy_ent *</code>	Output	The correctly formatted PHY Frames
<code>ahb_data</code>	<code>UINT8 *</code>	Input	The data from the AHB (the data to transmit)
<code>len_ahb</code>	<code>int</code>	Input	The length of <code>ahb_data</code>
<code>mgmt_data</code>	<code>UINT8 *</code>	Input	The MAC Management data
<code>len_mgmt</code>	<code>int</code>	Input	The length of the management data
<code>enc_key</code>	<code>UINT8</code>	Input	The encryption key
<code>cap</code>	<code>UINT8</code>	Input	Priority of the frame
<code>cc</code>	<code>UINT8</code>	Input	Contention Control bit
<code>resp_type</code>	<code>UINT8</code>	Input	Response frame required
<code>tm_data</code>	<code>tm *</code>	Input	Pointer to the Tone Map data structure
<code>num_frames</code>	<code>int *</code>	Output	The number of frames in the message

Table 5.1 – MAC Frame Assembler Inputs and Outputs

The algorithm for the function is given below:

1. Copy the source and destination addresses from the AHB data (source is the first 6 bytes, destination the second)
2. Copy the sequence number from the AHB data (next 2 bytes)
3. Get the Tone Map and its status
4. If the Tone Map is stale or invalid
 - a. Use the default (ROBO) Tone Map
5. Create the Service Block (`servb_create`)
6. Get the segmentation information (`segment`)
7. Create the frames (NB all frames are created, and the transmission of these to the PHY is controlled by the top-level MAC function).

5.2.1.1 Service Block Create (`servb_create`)

This function takes the information from the data passed to the MAC (along with the data to transmit) and creates the Service block, which is shown in Figure 5.2.

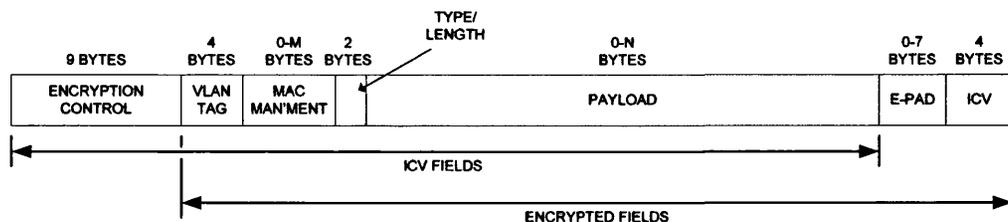


Figure 5.2 – Service Block

Certain fields (VLAN Tag, Type/Length and Payload) are passed to the MAC, and the rest are generated internally by the function. The inputs and outputs to the function are given in Table 5.2.

Variable Name	Type	Direction	Description
enc_servb	UINT8 *	Output	The complete and encrypted Service Block
data_in	UINT8 *	Input	The input data from the AHB
data_st	int	Input	The starting byte of the input data (as it also contains the addresses)
length	int	Inptu	The length of data_in
mgmt_data	UINT8 *	Input	The MAC Management data
mgmt_len	int	Inptu	The length of the management data
enc_key	UINT8	Input	The Encryption Key
epad_len	int *	Output	The length of the E-PAD

Table 5.2 – MAC Service Block Inputs and Outputs

The algorithm for the function is given below:

1. Is a VLAN Tag present (1st and 2nd byte is 0x8100)
 - a. Copy the VLAN to the Service Block
 - b. Set the start of the payload to the start of the data (i.e. the end of the addresses) plus the VLAN length
2. Else
 - a. Set the start of the payload to the start of the data
3. If the length of the MAC Management data is greater than 0
 - a. Add the MAC Management data to the Service Block
4. Add the rest of the data to the Service Block
5. Calculate the length of the E-PAD
6. Add the E-PAD to the Service Block
7. Calculate the Intergrity Check Value (ICV) (`calc_icv`)
8. Add the ICV to the Service Block
9. Encrypt the full Service Block (`encrypt`)
10. Return the length of the Service Block

5.2.1.2 Segmentation Information (**segment**)

This block calculates the number of segments needed to transmit the Service Block; along with the number of 20- and 40-Symbol PHY blocks in the last segment (all the others will have 4 40-Symbol PHY blocks). Other information returned is the size of the B-PAD and the number of bytes in a 20- and 40-Symbol PHY Block.

The inputs and outputs to the function are given in Table 5.3.

Variable Name	Type	Direction	Description
servb_len	int	Input	The length of the Service Block
num_carriers	int	Input	The number of carriers
modulation	modl	Input	The modulation scheme
code_rate	punct	Input	The convolutional code rate
output	seg_info	Output	Information about the segments (function return value)

Table 5.3 – Segmentation Information Inputs and Outputs

The algorithm for the function is given below:

1. Initialise (clear) the output structure.
2. Calculate the overhead associated with each frame (Segment Control length plus the source and destination address length plus the Frame Check Sequence length)
3. Calculate the number of bytes in a 20- and 40-Symbol PHY block (`block_bits`)
4. Calculate the maximum number of bytes in a frame (4 times the number of bytes in a 40-Symbol PHY block)
5. Determine the number of maximum length frames, and the number of segments
6. If there is data left, calculate the number of 20- and 40-Symbol PHY blocks
 - a. Increment the number of segments
 - b. While there is sufficient data for a 40-Symbol PHY
 - i. Increment the number of 40-Symbol PHY blocks
 - ii. Decrement the data left by the amount in a 40-Symbol PHY
 - c. If there is more data than would fit in a 20-Symbol PHY Block
 - i. Increment the number of 40-Symbol PHY Blocks
 - d. Else
 - i. Increment the number of 20-Symbol PHY Blocks
7. Determine the length of the B-PAD
8. Return the output structure

5.2.1.3 Frame Create (**frame_create**)

The Frame Creation function takes the Service Block, along with the segmentation information and creates the PHY frames needed to transmit the Service Block. The format of the frame is shown in Figure 5.3.

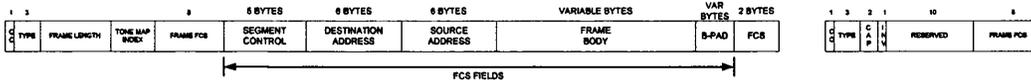


Figure 5.3 – PHY Frame Format

The function returns an array of structures containing the Start Frame Control, Payload and End Frame Control required to transmit the Service Block. The inputs and outputs to the function are given in Table 5.4.

Variable Name	Type	Direction	Description
frames	macphy_ent *	Output	The properly formatted PHY frames for transmission
segmentation	seg_info	Input	The segmentation information
serv_block	UINT8 *	Input	The Service Block to transmit
tmi	UINT8	Input	The Tone Map Index (for the SFC)
src_addr	UINT8 *	Input	The source MAC address
dest_addr	UINT8 *	Input	The destination MAC address
cap	UINT8	Input	Channel Access Priority
cc	UINT8	Input	Contention Control bit
resp_type	UINT8	Input	Response type (0=No response, 1=response)
epad_len	int	Input	Length of E-PAD
seq_num	int	Input	Sequence number for this Service Block

Table 5.4 – MAC Service Block Inputs and Outputs

The algorithm for the function is given below:

1. Initialise the variables
2. For each segment
 - a. Determine if this is the last segment
 - b. Create the Start Frame Control (SFC)
 - i. Copy the Contention Control bit to the SFC
 - ii. Create and copy the Delimiter Type to the SFC
 - iii. Determine the frame length from the number of 20- and 40-Symbol PHY blocks
 - iv. Copy the Frame Length to the SFC
 - v. Copy the Tone Map Index bits to the SFC
 - vi. Calculate the Frame Control Check Sequence (FCCS) (*calc_fccs*)
 - vii. Copy the FCCS to the SFC
 - c. Create the End Frame Control (EFC)
 - i. Copy the Contention Control bit to the EFC

- ii. Create and copy the Delimiter Type to the EFC
 - iii. Copy the frame priority to the EFC
 - iv. Fill the reserved fields with 0's
 - v. Calculate the FCCS (`calc_fccs`)
 - vi. Copy the FCCS to the EFC
- d. Create the payload
- i. Create the 5 bytes of the Segment Control Field and copy them to the frame
 - ii. Copy the destination address to the frame
 - iii. Copy the source address to the frame
 - iv. Copy the bytes of the message to the frame
 - v. If this is the last frame, copy the B-PAD
 - vi. Calculate the Frame Check Sequence (FCS) (`calc_fcs`)
 - vii. Copy the FCS to the frame
 - viii. Convert the bytes of the frame to bits, and store in the frame output array

5.2.2 MAC FRAME RE-ASSEMBLER (DISASSEMBLE)

The re-assembler function takes the frames from the PHY and re-creates the Service Block. It also checks the validity of each frame and creates the appropriate response frame. It is made up of three functions, which are given below.

1. Frame Checker (`frame_check`)
2. Response Frame Creations (`resp_create`)
3. Service Block Re-Assembler (`servb_recreate`)

The inputs and outputs to the function are given in Table 5.5.

Variable Name	Type	Direction	Description
<code>sfc</code>	UINT8 *	Input	Start Frame Control
<code>payload</code>	UINT8 *	Input	Frame payload
<code>efc</code>	UINT8 *	Input	End Frame Control
<code>pl_len</code>	int	Input	Payload length
<code>segs</code>	seg_blocks*	In/Out	Segments that make up the service block
<code>rfc</code>	UINT8 *	Output	Response Frame Control
<code>rx_servb</code>	UINT8 *	Output	The decoded Service Block
<code>servb_len</code>	int *	Output	The length of the service block

Table 5.5 – MAC Frame Re-Assembler Inputs and Outputs

The algorithm for the function is given below:

1. Check the validity of the received frame (`frame_check`)
2. Get the FCS from the received frame (for the response)
3. Get the priority of the frame
4. Generate the Response Frame Control (RFC) (`resp_create`)
5. If the frame is valid
 - a. Get the Segment Count from the payload Segment Control field
 - b. Copy the payload to the segment structure (holds the received frames)
 - c. Get the Last Segment Flag from the Segment Control field of the payload
 - d. If this is the last segment
 - i. Check if all the received frames are valid
 - ii. If they are, recreate the Service Block (`servb_recreate`), and set the return value to "Complete Service Block"
 - iii. If they are not, set the return value to "Incomplete Service Block"
 - e. Otherwise
 - i. set the return value to "Incomplete Service Block"
6. Return the status of the service block

5.2.2.1 Frame Checker (**frame_check**)

The frame check function takes the incoming frame and checks that the FCCS in the Start and End Frame Controls and the FCS in the Payload are valid (i.e. there are no errors in the frame). It also checks if any the fields within the frame are invalid. For example, maybe the Frame Length field is wrong, which could happen if the frame came from a device running a different version of the protocol (although at this time there is only Version 1.0 of the HomePlug standard available). The function will return a value indicating if the frame is valid (0). The inputs and outputs to the function are given in Table 5.6.

Variable Name	Type	Direction	Description
sfc	UINT8 *	Input	Start Frame Control
payload	UINT8 *	Input	Payload
efc	UINT8 *	Input	End Frame Control
pl_len	int	Input	Length of the payload

Table 5.6 – MAC Service Block Inputs and Outputs

The algorithm for the function is given below:

1. Calculate the FCS based on the received payload (*calc_fcs*)
2. Compare this with the received FCS. If they are different, mark the frame as invalid
3. Check the FCCS of the Start Frame Control (*calc_fccs*)
4. Compare this with the received FCCS. If they are different, mark the frame as invalid
5. Check the FCCS of the End Frame Control (*calc_fccs*)
6. Compare this with the received FCCS. If they are different, mark the frame as invalid
7. If the Tone Map Index (in the SFC) is greater than 0x10, the frame is invalid
8. If the Frame Length (in the SFC) is greater than 0x08, the frame is invalid
9. If the "Invalid" flag is set (in the EFC), the frame is invalid
10. If the Frame Protocol Version (in the Payload Segment Control) is not 0, the frame is invalid
11. Return the validity of the frame

5.2.2.2 Response Frame Create (**resp_create**)

The response frame creation function creates the appropriate Response Frame Control (RFC) based on the validity of the received frame. The inputs and outputs to the function are given in Table 5.7.

Variable Name	Type	Direction	Description
rfc	UINT8 *	Output	Response Frame Control
rfcs	UINT8 *	Inptu	Received Frame Check Sequence
cap	UINT8	Input	Channel priority
cc	UINT8	Input	Contention Control
resp_type	int	Input	Type of response (1=ACK, 0=NACK)

Table 5.7 – MAC Service Block Inputs and Outputs

The algorithm for the function is given below:

1. Copy the Contention Control bit to the RFC
2. Set the upper 2 bits of the Delimiter Type to 10 (response frame)
3. Copy the message priority
4. If the Response frame is valid
 - a. Set the last bit of the Delimiter Type to 0 (ACK Response)
 - b. Copy the received FCS to the RFC
5. Else
 - a. Set the last bit of the Delimiter Type to 1 (NACK/FAIL Response)
 - b. Set the Response Type to 0 (NACK Response)
 - c. Set the remaining bits to 0
6. Calculate the FCCS (*calc_fccs*)
7. Copy the FCCS to the RFC

5.2.2.3 Service Block Recreate (*servb_recreate*)

The final stage is to recreate the Service Block as it was transmitted, from the frames that make it up. This is shown in Figure 5.4.

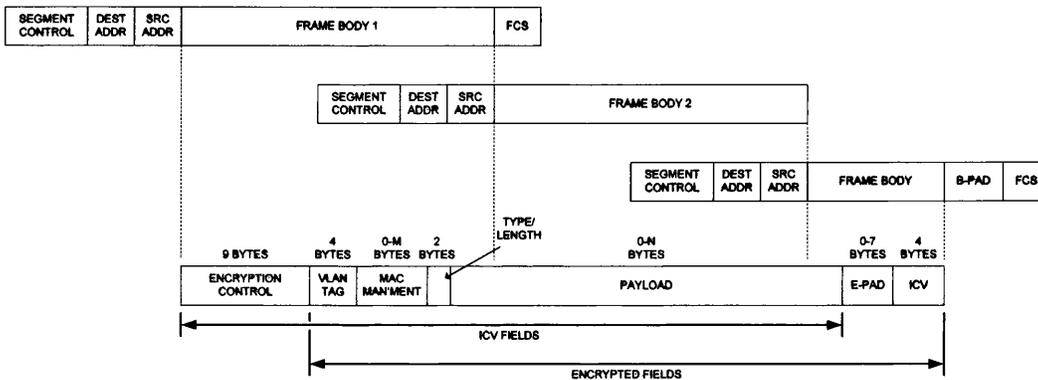


Figure 5.4 – Service Block Re-creation Process

The inputs and outputs to the function are given in Table 5.8.

Variable Name	Type	Direction	Description
segs	seg_blocks*	Input	The received segments
servb	UINT8 *	Output	The re-created service block

Table 5.8 – MAC Service Block Recreation Inputs and Outputs

The algorithm for the function is given below:

1. Initialise the pointers
2. While the segment isn't the last
 - a. Copy the payload to the Service Block
 - b. Increment the segment number
3. Return the length of the Service Block

5.3 PHY MODEL

Figure 5.5 shows the components that make up the HomePlug functionality of the Physical Layer model. These are the non-threaded components (except for the top level PHY block) and can be used as stand-alone functions.

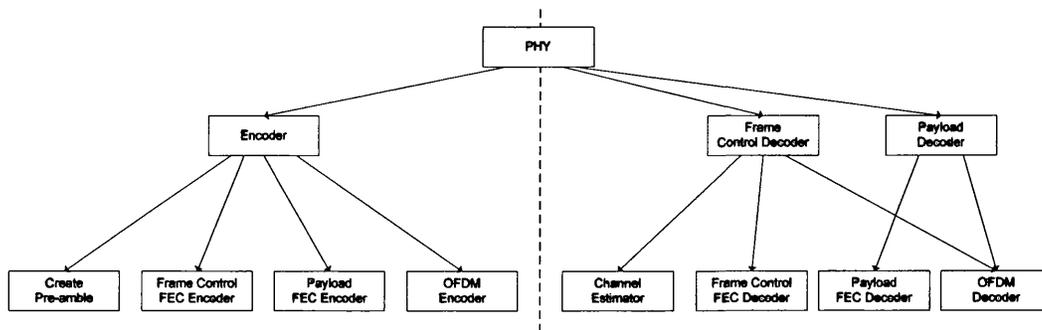


Figure 5.5 – PHY Structure Chart

The diagram is split into the transmitter function (Encoder) and the receiver functions (Frame Control Decoder and Payload Decoder). The decoder is split into two sections so the block can process the Frame Control separately. This is needed as the block communicates information in the Frame Control (namely the Tone Map Index) to get the correct information with which to decode the Payload. The transmitter and receiver functions are described below and the diagram above is expanded to give further details. The algorithms for each block (and their sub-blocks) are also given.

5.3.1 PHY ENCODER

The PHY Encoder takes the Start Frame Control, Payload and End Frame Control from the MAC and encodes the data for transmission on the channel. Figure 5.6 gives the structure chart that shows the sub-functions that make up the encoder. Figure 5.7 shows the data flow through these blocks.

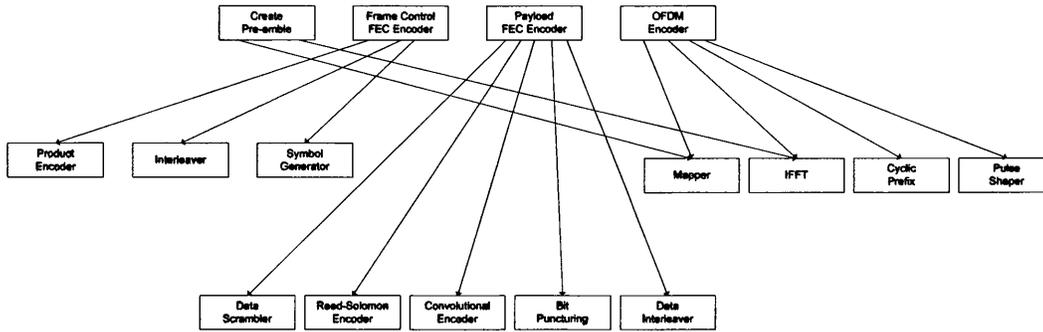


Figure 5.6 – PHY Encoder Structure Chart

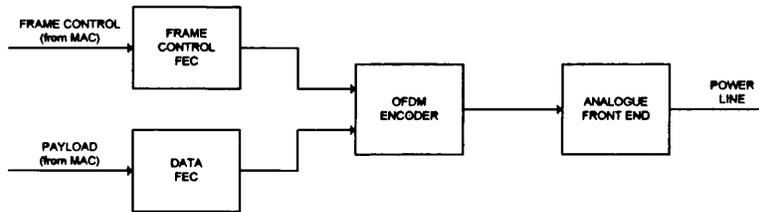


Figure 5.7 – PHY Encoder Data Flow

The inputs and outputs to the encoder are given in Table 5.9.

Variable Name	Type	Direction	Description
phy_data	float *	Output	Signal to transmit on the channel
sfc	UINT8 *	Input	Start Frame Control (25 bits)
efc	UINT8 *	Input	End Frame Control (25 bits) or NULL
payload	UINT8 *	Input	Payload (variable bits)
pl_len	int	Input	The length of the payload (in bits)
tonemask	UINT8 *	Input	The system wide Tone Mask
tonemap	tm_data	Input	The Tone Map to use for this transmission
phase	float *	Input	The reference phase for the OFDM encoder
prefix	int	Input	The length of the cyclic prefix
dest_dir	char *	Input	Results directory to store the encoded data in.

Table 5.9 – PHY Encoder Inputs and Outputs

If the encoder is transmitting a response frame, then the efc and payload inputs will be NULL and the payload length zero. The function will determine the frame type from the Frame Type field in the Start Frame Control.

The algorithm for the encoder is given below:

8. Determine the type of frame from the Start Frame Control
9. Calculate the number of carriers for the Frame Control (from the Tonemask)
10. Create the preamble (`pream_cre`)
11. Encode the Start/Response Frame Control through the Frame Control FEC (`fc_encode`)
12. Encode the Frame Control FEC output through the OFDM encoder (`ofdm_encode`)
13. If the frame is a Data frame (Frame Type in SFC = 3'b000 or 3'b001)
 - a. Calculate the number of usable carriers for the payload
 - b. Get the number of 40 and 20 Symbol PHY blocks from the Frame Length (`decode_fl`)
 - c. Encode the Payload through the Data FEC (`data_encode`)
 - d. Encode the Data FEC output through the OFDM encoder (`ofdm_encode`)
 - e. Encode the End Frame Control through the Frame Control FEC (`fc_encode`)
 - f. Encode the Frame Control FEC output through the OFDM encoder (`ofdm_encode`)

After each encode phase, the output will be stored in the directory given as an input, unless the directory is NULL. The output from the OFDM encoding is also copied to “phy_data” and at the end of the function this will contain the complete signal that will be transmitted on the channel (the transmission of which is controlled by the thread part of the PHY). The functions `fc_encode`, `data_encode` and `ofdm_encode` implement the functionality described in Sections 3.3.3.1 through 3.3.3.3 respectively, and the implementation of each is described in the following three sections.

5.3.2 FRAME CONTROL FEC ENCODER (fc_encode)

The Frame Control FEC is the implementation of the description given in Section 3.3.3.1 and is shown again in Figure 5.8. It consists of three sub functions:

1. Product Encoder (fc_proden)
2. Frame Control Interleaver (fc_inter)
3. Symbol Generator (fc_symgen)

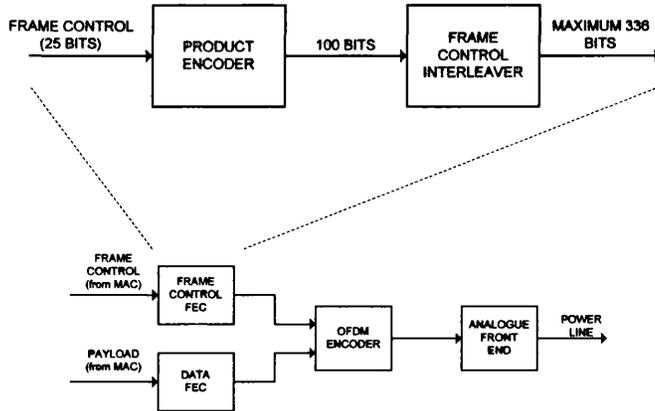


Figure 5.8 – Frame Control FEC Block diagram

The first two sub functions correspond directly to the description given in Section 3.3.3.3 and the last function ensures the interleaved data is placed on the correct bit positions on each of the 4 OFDM symbols that make up the Frame Control on the channel. In the HomePlug specification this is part of the interleaver function, however to give orthogonality with the decoder, it was separated into a function of its own. The inputs and outputs of the function are given in Table 5.10.

Variable Name	Type	Direction	Description
output	UINT8 *	Output	The encoded data stream, ready for encoding via the OFDM encoder
input	UINT8 *	Input	The 25-bits Frame Control from the MAC
num_car	int	Input	The number of usable carriers

Table 5.10 – Frame Control FEC Input and Outputs

The algorithm for the function is simply to pass the input data through each of the functions in turn. This gives the required number of encoded bits for the OFDM encoder. Each of the functions is now described.

5.3.2.1 Product Encoder (fc_proden)

The Product Encoder takes the 25 bits of the Frame Control (from the MAC) and generates 100 output bits that consist of the column and row parity of the input data. Table 5.11 gives the input and output from the function.

Variable Name	Type	Direction	Description
prod_data	UINT8 *	Output	The 100 Encoded output bits
data	UINT8 *	Input	The 25-bits Frame Control from the MAC

Table 5.11 – Product Encoder Input and Outputs

The function uses a 100-element array to represent the [10x10] matrix used in the product encoding algorithm. Figure 5.9 shows the indexes of the matrix (starting at 1) and how these correspond to the position within the matrix, as well as what groups correspond to the theoretical description given in Chapter 3.

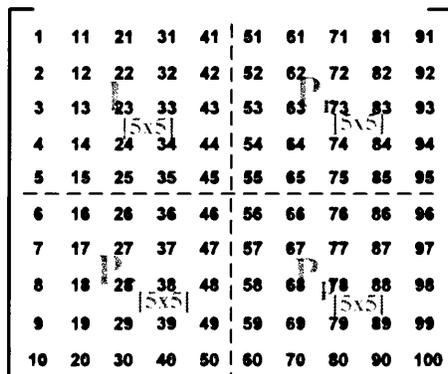


Figure 5.9 – Product Encoder Matrix

The algorithm for the function is given below.

1. Copy the input data to position "I" (as shown in the diagram above)
2. For each column of data (the first 5 at this time), calculate the parity and place the results in position "P_c".
3. For each row of data (there are 10 now), calculate the parity and place the results in position "P_r" (for the position "I" data) and in position "P_p" (for position "P_c" data)

The function used for calculating the parity is a set of exclusive-or's. These are shown diagrammatically in Figure 5.10. They show the input bits which are exclusive-or'd together to create each parity bit (the circle on the intersection between the horizontal and vertical lines).

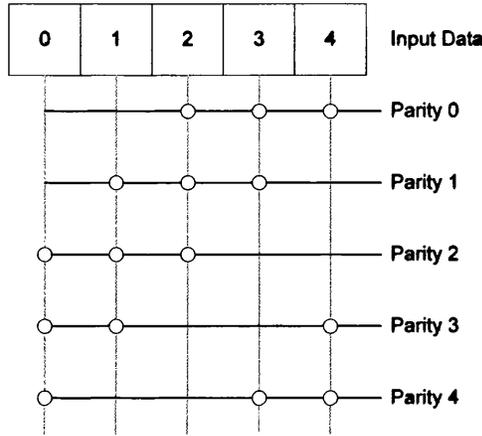


Figure 5.10 – Product Encoder Parity Generation

5.3.2.2 Frame Control Interleaver (**fc_inter**)

The next function is the Frame Control Interleaver. This takes the 100 bits from the product encoder and applies the interleaving algorithm given in the HomePlug specification. Table 5.12 gives the input and output from the function.

Variable Name	Type	Direction	Description
output	UINT8 *	Output	The 100 interleaved bits
data	UINT8 *	Input	The 100 bits from the product encoder

Table 5.12 – Frame Control Interleaver Input and Outputs

The algorithm used is from the HomePlug Specification [62] and as implemented in the model. This ensures the data is sufficiently “random”.

5.3.2.3 Frame Control Symbol Generator (**fc_symgen**)

The final stage of the Frame Control FEC is to place the 100 bits from the interleaver over the four OFDM symbols. The process is to place the interleave bits sequentially over the four symbols, excluding unused carriers, and applying a shift of 25 bits at the end of each symbol. This is shown in Figure 5.11. The indexes are those for the default HomePlug Tonemask.

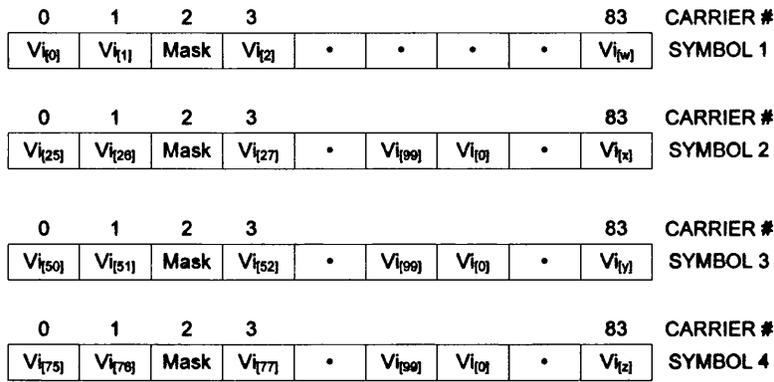


Figure 5.11 – Symbol Generator Process

5.3.3 PAYLOAD FEC ENCODER (DATA_ENCODE)

The Payload FEC encoder is the implementation of the description given in Section 3.3.3.2 and the block diagram is shown again in Figure 5.12. It consists of five sub functions

1. Data Scrambler (scramble)
2. Reed Solomon Encoder (rs_enc)
3. Convolutional Encoder (conv_enc)
4. Bit Puncturing (puncture)
5. Data Interleaver (interleave)

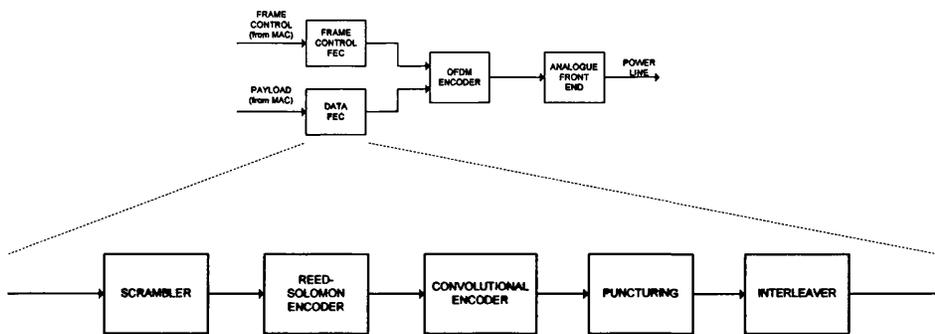


Figure 5.12 – Payload FEC Encoder Block Diagram

The inputs and outputs to the function are given in Table 5.13.

Variable Name	Type	Direction	Description
enc_pl	UINT8 *	Output	The encoded payload
payload	UINT8 *	Input	The complete payload to encode
mod	mod1	Input	The modulation scheme to use
mode	punct	Input	Half or Three quarter rate puncturing
num_car	int	Input	The number of usable carriers
length	int	Input	The length of the payload (in bits)
num20	int	Input	The number of 20 Symbol PHY blocks
num40	int	Input	The number of 40 Symbol PHY blocks
dir	char *	Input	The directory to store the results in

Table 5.13 – Payload FEC Inputs and Outputs

The algorithm isn't quite as simple as that of the Frame Control FEC encoder, as depending on the size of the payload and modulation scheme used the functions might be called repeatedly. This happens if there is more than one 40-symbol PHY block. Also the Reed-Solomon Encoder is configured differently when ROBO modulation is used. The algorithm is:

1. Calculate the number of bits in a 20- and 40-symbol PHY block, plus the size and number of Reed-Solomon blocks (using the `block_bits` function)
2. Read the Reed-Solomon generator functions (from file) for the modulation mode.
3. For each 40-Symbol PHY block
 - a. Get the input bits that make this PHY block
 - b. Scramble the data from 3a (`scramble`)
 - c. Save the results of the Scrambler (see algorithm below)
 - d. If there is only one RS block
 - i. Encode the scrambled data through the Reed-Solomon Encoder (`rs_enc`).
 - ii. Calculate the length of the encoded data
 - e. Otherwise
 - i. Calculate the length of the encoded data
 - ii. For each Reed-Solomon block
 1. Calculate the number of bits in the block (this is either a full Reed-Solomon block or the remainder)
 2. Extract the number of bits in the block from the Scrambled data
 3. Encode the bits from the step above through the Reed-Solomon Encoder (`rs_enc`)
 - f. Save the results
 - g. If using half-rate convolutional encoding
 - i. Pass the Reed-Solomon encoded data through the convolutional encoder (`conv_enc`) and calculate the length of the resulting data.
 - ii. Save the results
 - h. else
 - i. Pass the Reed-Solomon encoded data through the convolutional encoder (`conv_enc`)
 - ii. Pass the encoded data through the bit puncturer (`puncture`). This function returns the length of the output
 - iii. Save the results
 - i. Interleave the data through the Data Interleaver (`interleave`)
 - j. Save the results
 - k. Copy the fully encoded PHY block to the output
4. If there is a 20 Symbol PHY
 - a. Get the input bits that make this PHY block
 - b. Scramble the data from 3a (`scramble`)
 - c. Save the results of the Scrambler (see algorithm below)
 - d. If there is only one RS block

- i. Encode the scrambled data through the Reed-Solomon Encoder (`rs_enc`).
 - ii. Calculate the length of the encoded data
- e. Otherwise
 - i. Calculate the length of the encoded data
 - ii. For each Reed-Solomon block
 1. Calculate the number of bits in the block (this is either a full Reed-Solomon block or the remainder)
 2. Extract the number of bits in the block from the Scrambled data
 3. Encode the bits from the step above through the Reed-Solomon Encoder (`rs_enc`)
- f. Save the results
- g. If using half-rate convolutional encoding
 - i. Pass the Reed-Solomon encoded data through the convolutional encoder (`conv_enc`) and calculate the length of the resulting data.
 - ii. Save the results
- h. else
 - i. Pass the Reed-Solomon encoded data through the convolutional encoder (`conv_enc`)
 - ii. Pass the encoded data through the bit puncturer (`puncture`). This function returns the length of the output
 - iii. Save the results
- i. Interleave the data through the Data Interleaver (`interleave`)
- j. Save the results
- k. Copy the fully encoded PHY block to the output

As noted in the algorithm, the save results is not a simple case of outputting to a file. The algorithm used is

1. If the directory to save in is not NULL
 - a. Create the full file name (appropriate to the stage of the algorithm)
 - b. If this is the first time through
 - i. Set the access mode to open and over-write
 - c. else
 - i. Set the access mode to open and append
 - d. Open the file with the correct name and access mechanism
 - e. If it fails to open, report an error

- f. Otherwise print the output of the current stage of the main algorithm, and a separator

Each of the functions that make up the Payload FEC Encoder is described below.

5.3.3.1 Data Scrambler (**scramble**)

The first block is the scrambler. This ensures that there are no runs of ones or zeros which could happen where there is zero padding in the payload for example. Runs of ones and zeros reduce the effectiveness of the error correcting algorithms. The scrambler function is an implementation of the circuit shown in Figure 5.13.

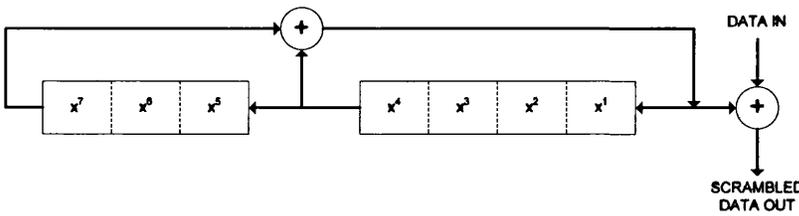


Figure 5.13 – Scrambler Circuit

The inputs and outputs of the function are given in Table 5.14.

Variable Name	Type	Direction	Description
data	UINT8 *	In/Out	The data stream to be scrambled
length	int	Input	The length of the data (in bits)

Table 5.14 – Scrambler Function Inputs and Outputs

The algorithm of the function is

1. Initialise the pseudo random sequence to all ones
2. For each bit in the data
 - a. Calculate the exclusive-or value ($x^7 \oplus x^4$)
 - b. Shift the sequence to the left (ie $x^7 = x^6$, etc)
 - c. Store the exclusive-or value in x^1
 - d. Set the output value equal to the input value exclusive-or'd with the input value

5.3.3.2 Reed Solomon Encoder (*rs_enc*)

The second function the data passes through is the Reed-Solomon Encoder, which encodes the data using the Reed-Solomon algorithm [78]. This adds parity to the end of the data. The number of parity symbols generated depends on the modulation scheme used. If it is DQPSK or DBPSK then there are 16 parity symbols and if the modulation is ROBO then there are 8 parity symbols. The function implements the Reed-Solomon Encoder shown in Figure 5.14.

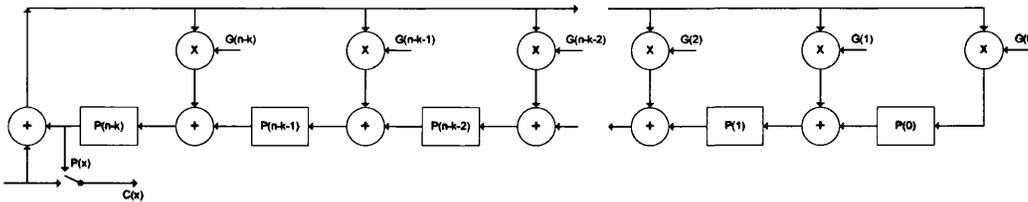


Figure 5.14 – Reed Solomon Encoder Circuit

The inputs and outputs of the function are given in Table 5.15.

Variable Name	Type	Direction	Description
msg	UINT8 *	In/Out	The input message (binary) – the parity is appended to this
mod	mod1	Input	The modulation
length	int	Input	The length of msg (in bits)
gen	int *	Input	The generator polynomial to encode the data

Table 5.15 –Reed-Solomon Encoder Function Inputs and Outputs

The algorithm of the function is

1. Determine the number of parity symbols (8 if ROBO modulation, 16 if DQPSK or DBPSK)
2. Create the arrays to convert between power and tuple form (*fieldgen*)
3. Convert the input into Reed-Solomon symbols in power form (8 bits per symbol). Uses the *tup2pow* array from the previous step.
4. Clear the parity array
5. For each symbol in the message
 - a. Calculate the feedback value (current symbol \oplus last parity symbol)
 - b. For each parity symbol
 - i. Calculate the addition of the feedback value and the generator coefficient
 - ii. Calculate the new value of the parity symbol (addition from step above \oplus the previous parity symbol)
6. Convert the parity symbols back to binary and append to the input message.

5.3.3.3 Convolutional Encoder (conv_enc)

The next stage is the convolutional encoder. This takes the data from the Reed-Solomon encoder and encodes the data using the circuit in Figure 5.15. The function (conv_enc) is the implementation of this circuit. This generates double the amount of output data as input; hence it is a half-rate code.

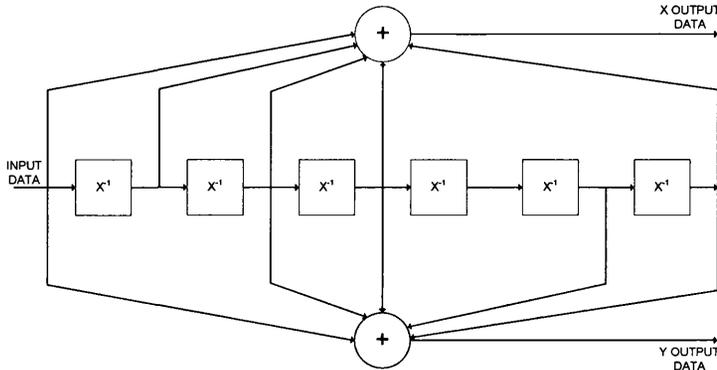


Figure 5.15 – Convolutional Encoder Circuit

The inputs and outputs of the function are given in Table 5.16.

Variable Name	Type	Direction	Description
output	UINT8 *	Output	Output data stream (interleaved X-Y)
input	UINT8 *	Input	Input data stream
length	int	Input	Length of input stream

Table 5.16 – Convolutional Encoder Inputs and Outputs

The algorithm used by the function is

1. Initialise the shift register (set all elements to zero)
2. For each bit in the input
 - a. Calculate the X stream output ($\text{input} \oplus \text{shift}[0] \oplus \text{shift}[1] \oplus \text{shift}[2] \oplus \text{shift}[5]$)
 - b. Calculate the Y stream output ($\text{input} \oplus \text{shift}[1] \oplus \text{shift}[2] \oplus \text{shift}[4] \oplus \text{shift}[5]$)
 - c. Store the X and Y values in the output (interleaving them so that the output is X-Y-X-Y)
 - d. Update the shift register ($\text{shift}[x] = \text{shift}[x-1]$), and store the input bit in $\text{shift}[0]$
3. Flush the shift register with zeros (this returns it to a known state for decode and uses the same algorithm as step 2)

5.3.3.4 Bit Puncturing (puncture)

After the data has been through the convolutional encoder, it is optionally passed through the bit puncturer. This makes the half-rate data created in the convolutional encoder three-quarter rate by removing the bits according to the pattern in Figure 5.16.

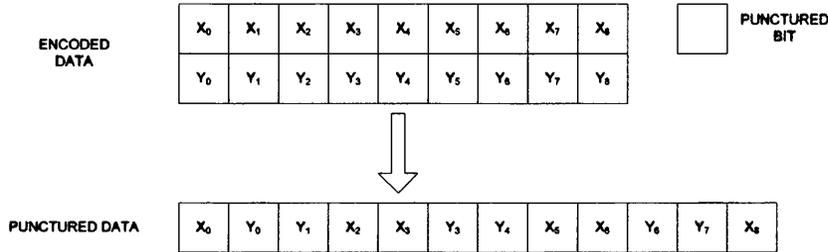


Figure 5.16 – Bit Puncturing

The inputs and outputs for the function are given in Table 5.17.

Variable Name	Type	Direction	Description
output	UINT8 *	Output	Punctured data stream
input	UINT8 *	Input	Input data stream (contains X & Y)
mode	punct	Input	Puncture Rate ($\frac{1}{2}$ or $\frac{3}{4}$ rate)
len_in	int	Input	Length of input stream (in bits)

Table 5.17 – Bit Puncture Inputs and Outputs

The algorithm used by the block is given below

1. For each input bit
 - a. If the bit is not punctured (i.e. is not a multiple of 3)
 - i. Copy the input bit to the output
 - ii. Increment the output stream index
2. Return the output stream index (this gives the length of the output)

5.3.3.5 Data Interleaver (interleave)

The final stage is the interleaver. This ensures that the logically adjacent data isn't transmitted physically adjacent, and also introduces the extra redundancy when ROBO mode is used. The interleaver itself is a simple row/column interleaver, which populates a matrix by filling it column wise and reading out the data row wise. This is shown in Figure 5.17. The function is an implementation of this.

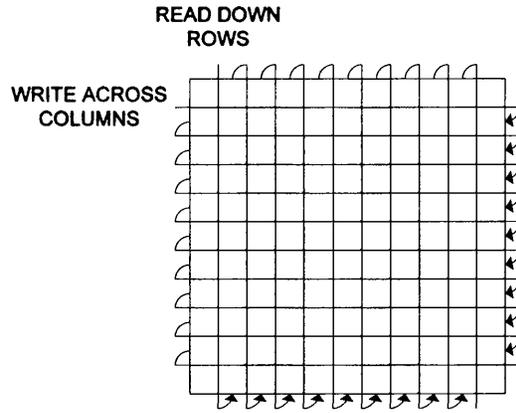


Figure 5.17 – Interleaver Process

The inputs and outputs of the function are given in Table 5.18.

Variable Name	Type	Direction	Description
data	UINT8 *	In/Out	Data stream to be interleaved
mod	mod1	Input	Modulation scheme being used
len	int	Input	Length of the input data stream
num_car	int	Input	Number of carriers that can be used
block_size	int	Input	Size of the PHY block that is being created – either 20 or 40

Table 5.18 – Interleaver Inputs and Outputs

The algorithm used is given below.

1. Set the parameters for the interleaver, depending on the modulation
 - a. ROBO – Number of rows is number of carriers, number of columns is 10 and number of symbols 4
 - b. DQPSK or DBPSK – Number of rows is twice number of carriers, number of columns is 10 or 20 (for a 20 or 40 symbol PHY block respectively) and the number of symbols is 1
2. Calculate the size of the output (number of symbols * number of rows * number of columns)
3. For each row (as determined in step 1)
 - a. Set the initial row index
 - b. For each column (as determined in step 1)
 - i. If the modulation is DQPSK, combine the bits from the “2” interleavers ($2 * \text{data}[\text{current input, second matrix}] + \text{data}[\text{current input, first matrix}]$). Place this in the interleaver at $[\text{current row}][\text{current column}]$
 - ii. If the modulation is DBPSK or ROBO, place the current data in the interleaver at $[\text{current row}][\text{current column}]$
 - iii. Increment the current input

- iv. Update the current row (8 less than previous), wrapping round if this is less than zero
4. Copy the interleaver to the output, creating the redundancy if using ROBO modulation.

The functions above will create the encoded version of the input data stream from the MAC and also ensure that there is sufficient data for the OFDM encoder.

5.3.4 OFDM ENCODER (ofdm_encode)

The OFDM Encoder function (ofdm_encode) is the implementation of the description given in Section 3.3.3.3. The block diagram of the function is given again in Figure 5.18, and the function consists of four main sub-functions

1. Data Mapper (mapper)
2. Inverse Fast Fourier Transform (ifft_block)
3. Cyclic extender (cyclic)
4. Pulse Shaper (pulse_shape)

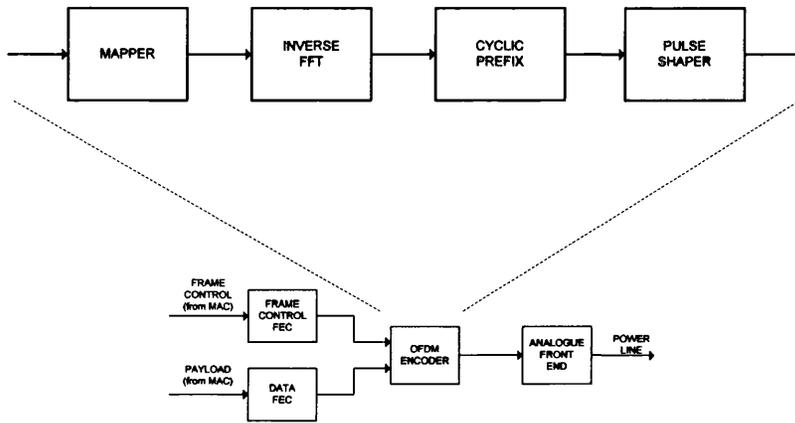


Figure 5.18 – OFDM Encoder

The inputs and outputs of the function are given in Table 5.19.

Variable Name	Type	Direction	Description
output	float *	Output	The Output data stream
data	UINT8 *	Input	The binary input stream
tonemask	UINT8 *	Input	The system-wide Tone Mask
tonemap	UINT8 *	Input	The link specific Tone Map
modulation	mod1	Input	The modulation scheme to use
phase	float *	In/Out	The reference phase (gets updated)
num_syms	int	Input	The number of symbols
num_car	int	Input	The number of usable carriers
dir	char *	Input	The results directory

Table 5.19 – OFDM Encoder Inputs and Outputs

The algorithm of the function is given below.

1. While there are still symbols left to encode
 - a. Determine the size of this PHY block (either 40 if there are more than 40 symbols left, or whatever is left)
 - b. Calculate the number of bits required for the PHY block and copy these from the input (number of bits = block size * number of carriers)

- c. Save this if requested
- d. Calculate the number of symbols left
- e. Pass the data from (b) through the Mapper
- f. Save the mapped data if requested
- g. Pass the mapped data from (e) through the IFFT
- h. Save this if requested
- i. Add the cyclic prefix
- j. Save this if requested
- k. Apply pulse shaping
- l. Save this if requested
- m. Copy the OFDM encoded data to the output

The “Save if requested” steps in the algorithm above use the same algorithm as the similar steps in the Payload FEC encoder.

5.3.4.1 Data Mapper (**mapper**)

The first block in the OFDM encoding process is the mapper. This takes the binary data from either the Frame Control FEC or Payload FEC and maps them to the correct phase and amplitude information (in the complex plane) for the IFFT. The function uses the constellations given in Figure 5.19, and it also adds in a reference phase¹. If the data is from the Frame Control FEC, this phase comes from Table 11 in the HomePlug Specification, and if the data is from the Payload FEC the reference phase is the phase of the data sent on that sub-carrier on the previous OFDM symbol, with the first symbol taking its reference phase from the last Frame Control symbol. This is shown in Figure 5.20.

¹ The inputs refer to the values in the interleaver matrix. In the case of DQPSK, they refer to the two interleaver matrices, although the interleaver output is the decimal equivalent of the two binary numbers so in the implementation of the mapper, the values come from only one matrix.

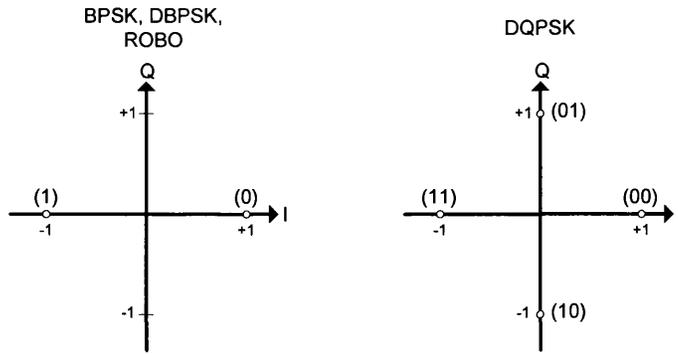


Figure 5.19 – Modulation Constellations

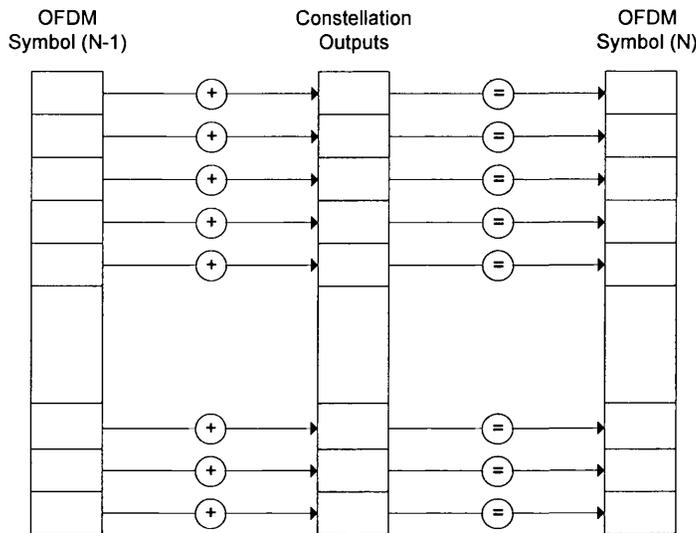


Figure 5.20 – Mapper Operation

The inputs and outputs of the mapper are given in Table 5.20.

Variable Name	Type	Direction	Description
output	complex *	Output	Mapped output
input	UINT8 *	Input	Input data stream (binary)
tonemask	UINT8 *	Input	System wide Tone Mask
tonemap	UINT8 *	Input	Link-specific Tone Map
mod	mod1	Input	Modulation scheme
num_syms	int	Input	Number of symbols in frame
phase	float *	In/Out	Reference Phase (gets updated)

Table 5.20 – Mapper Inputs and Outputs

The algorithm of the function is given below

1. Create the map to convert from the “binary” to the initial phase
2. Create the Pseudo Random array that is used when carriers are blocked from the tone map (pngen)
3. For each symbol

- a. If the carrier is not blocked via the tone mask
 - i. If the carrier is blocked via the tone map and the modulation is DQPSK or DBPSK then use the corresponding value in the Psuedo Random array to generate the phase, otherwise use the value in the input array to generate the phase (using the map generated in step 1)
- b. Else
 - i. This frequency is blocked, so set the phase to 0
- c. Add the reference phase
- d. Update the reference phase array if the data is payload or the last symbol of the Frame Control
- e. Calculate the real and imaginary components from the phase information (ie convert from a polar representation to a Cartesian representation)

The output of the mapper is the real and imaginary components which represent the data to be transmitted on each sub carrier. This is passed to the next stage, which is the IFFT Block.

5.3.4.2 Inverse Fast Fourier Transform (`ifft_block`)

The IFFT takes its input from the mapper, and generates the signal that will be transmitted on the channel. The function takes the data, and places it in the correct frequency “bins”, before creating the “negative frequencies”. This is shown in Figure 5.21. By creating the negative frequencies (which have the same real part, but inverse imaginary part) the output of the IFFT is purely real.

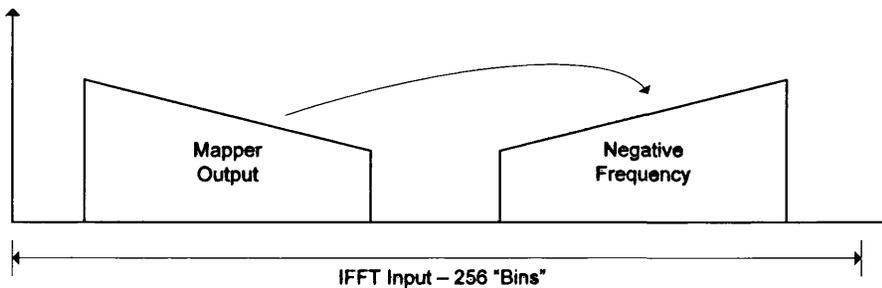


Figure 5.21 – IFFT Operation

The inputs and outputs of the function are given in Table 5.21.

Variable Name	Type	Direction	Description
output	float *	Output	Output data stream (waveform to transmit)
signal	complex *	Input	The input data (from mapper)
tonemask	UINT8 *	Input	The system-wide Tone Mask
sym	int	Input	The number of symbols in “signal”

Table 5.21 – IFFT Inputs and Outputs

The algorithm of the function is given below

1. For each symbol
 - a. Clear the IFFT Input array
 - b. Copy the input symbol to the correct frequency bins (23 to 107)
 - c. Create the negative frequency
 - d. Perform the IFFT (`ifft`)
 - e. Copy the real IFFT component to the output array

The function `ifft` is an implementation of the Inverse Fast Fourier Transform function.

5.3.4.3 Cyclic Prefix (`cyclic`)

After the data has been converted to the time domain, a cyclic prefix is added. This appends the last 172 samples of the symbol to the beginning of it, so that the 256 samples per symbol out of the IFFT become 428 samples per symbol. Figure 5.22 shows this diagrammatically.

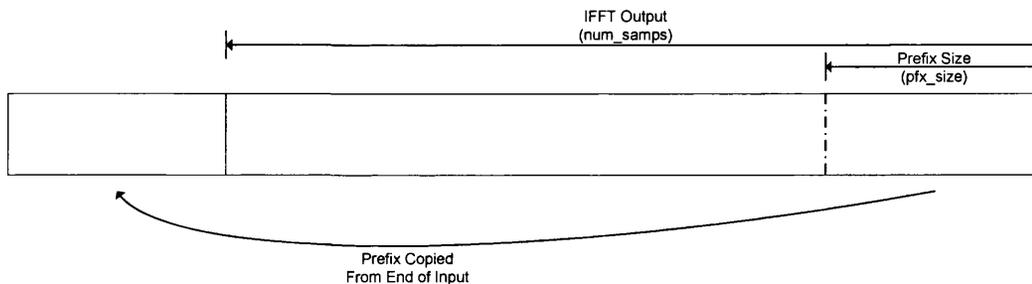


Figure 5.22 – Cyclic Prefix

The inputs and outputs of the function are given in Table 5.22.

Variable Name	Type	Direction	Description
output	float *	Output	Cyclically extended version of data
data	float *	Input	Input data (output from IFFT)
pfx_size	int	Input	The size of the prefix to add
num_samp	int	Input	The number of samples in each symbol
num_sym	int	Input	The number of symbols

Table 5.22 – Cyclic Prefix Inputs and Outputs

The algorithm of the function is given below

1. For each symbol
 - a. Copy the prefix from the last samples to the front
 - b. Copy the rest of the symbol to the output

5.3.4.4 Pulse Shaper

The final stage of the OFDM Encoder is the Pulse Shaper. This applies a raised cosine shape to each symbol, as shown in Figure 5.23.

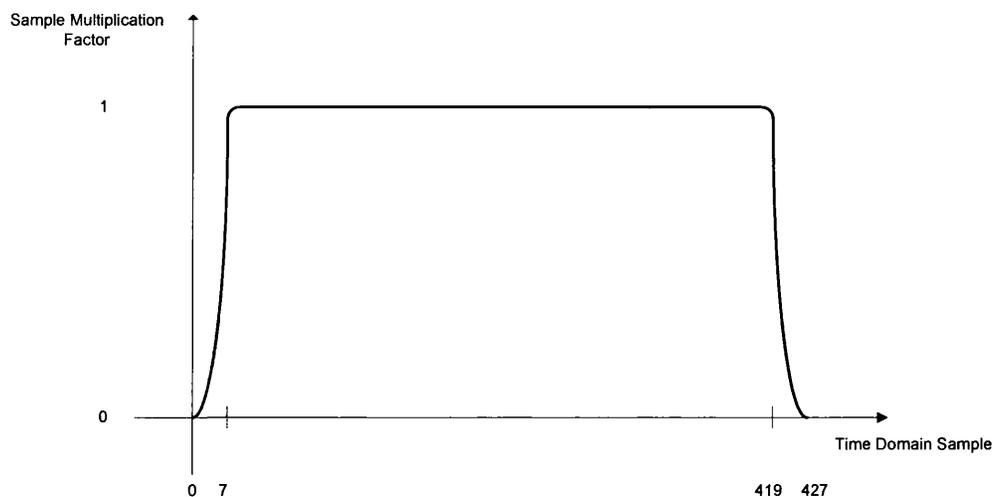


Figure 5.23 – Pulse Shaper

The inputs and outputs of the function are given in Table 5.23.

Variable Name	Type	Direction	Description
data	float *	In/Out	Symbols to transmit on the channel
num_syms	int	Input	The number of symbols in the data
num_samp	int	Input	The number of samples in each symbol
N	int	Input	The number of samples at the start and end of each symbol to shape

Table 5.23 – Cyclic Prefix Inputs and Outputs

The algorithm of the function is given below

1. For each symbol
 - a. For the first “N” samples
 - i. Shape the sample using the raised cosine function
 - b. For the last “N” samples
 - i. Shape the sample using the raised cosine function

5.3.5 PHY DECODER

The PHY Decoder is the inverse of the encoder. It takes the data from the channel (which will have been altered by the characteristics of the channel) and recreates the binary of the frame that was transmitted. This could involve correcting any errors that have occurred during transmission. The decoder is in fact made up of two separate functions

1. Frame Control Decoder (fc_dec)
2. Payload Decoder (pl_dec)

Figure 5.24 gives the structure chart that shows the functions that make up the decoder functions. Fig 6.25 shows the data flow through the blocks. Note that they use the same OFDM decoder.

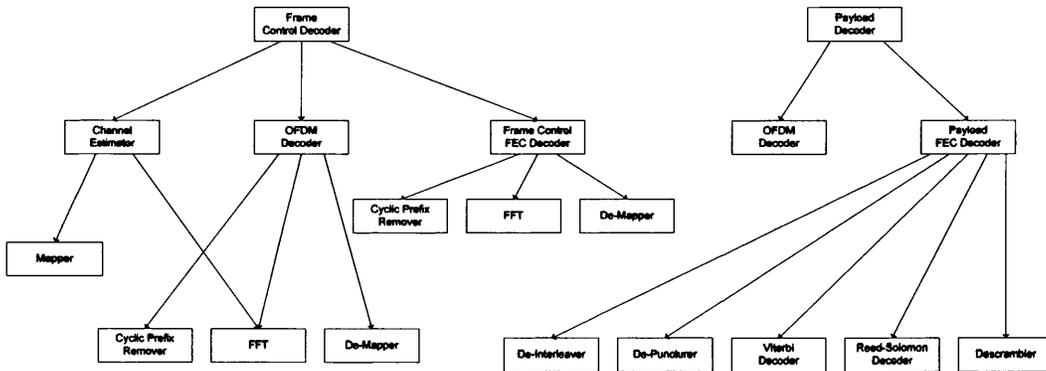


Figure 5.24 – PHY Decoder Structure Chart

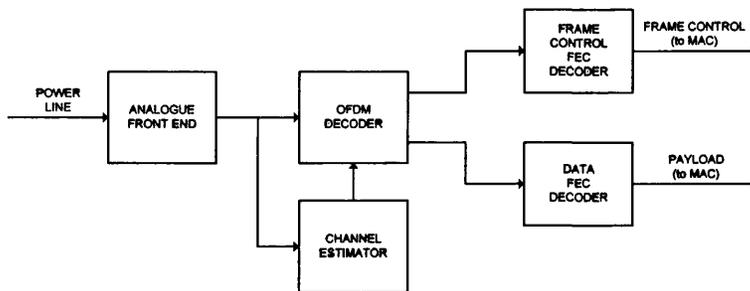


Figure 5.25 – PHY Decoder Data Flow

The reason there are two functions (as opposed to the one of the encoder) is because the payload decoder needs information from the Start Frame Control (the Tone Map Index) to get the number of carriers, modulation and code rate used to encode the data. This

information is part of the MAC functionality so the functions are split to allow the PHY to request this from the MAC (via the message protocol described previously). There are similarities between the two functions as they both use the same OFDM Decoder, however the Frame Control Decoder has a Frame Control FEC Decoder and the Payload Decoder a Payload FEC Decoder (which are the inverse of the FEC's in the encoder). The Frame Control decoder also has the channel estimation function in it, which determines the filter coefficients that are part of the OFDM decoder, and remove many of the channel characteristics from the received signal (this is an advantage of OFDM). The inputs and outputs of the Frame Control Decoder and Payload Decoder are given in Tables 5.24 and 5.25 respectively.

Variable Name	Type	Direction	Description
dec_fc	UINT8 *	Output	The decoded version of the Frame Control
fc	float *	Input	The received Frame Control (from the channel)
pream	float *	Input	The received Preamble
tonemask	UINT8 *	Input	The system-wide Tone Mask
phase	float *	In/Out	The reference phase (gets updated)
coeff	complex *	Output	The channel filter coefficients

Table 5.24 – Frame Control Decoder Inputs and Outputs

Variable Name	Type	Direction	Description
dec_pl	UINT8 *	Output	The decoded version of the Payload
pl	float *	Input	The received Payload (from the channel)
sfc	UINT8 *	Input	The Start Frame Control
tonemask	UINT8 *	Input	The system-wide Tone Mask
tonemap	tm_data	Input	The Tone Map for the transmission
phase	float *	Input	The reference phase
coeff	complex *	Input	The channel filter coefficients

Table 5.25 – Payload Decoder Inputs and Outputs

The Frame Control Decoder algorithm is

1. Use the preamble to determine the channel filter coefficients
2. Calculate the number of carriers from the Tone Mask
3. Decode the frame control through the OFDM decoder (`ofdm_decode`)
4. Decode the bits from the OFDM decoder through the Frame Control FEC Decoder (`fc_decode`)
5. Return the type of Frame Control (`dec_fc`)

The Payload Decoder algorithm is

1. Calculate the number of carriers (from the Tone Mask and Tone Map)
2. Get the number of 20 and 40 symbol PHY blocks (`decode_fl`)
3. Pass the received message through the OFDM decoder (`ofdm_decode`)

4. Pass the output from the OFDM decoder through the Payload FEC Decoder (data_decode)
5. Return the length of the payload

5.3.5.1 Channel Estimation (**chan_est**)

The first stage of the decoding process is channel estimation. This uses the received preamble to estimate the conditions of the channel and from this the filter coefficients which are used on the actual data (Frame Control and Payload). This negates much of the changes in amplitude and phase introduced by the channel. The inputs and outputs are given in Table 5.26.

Variable Name	Type	Direction	Description
cor	complex *	Output	The channel filter factors
rx_pream	float *	Input	The received preamble
phase	float *	Input	The reference phase
tonemask	UINT8 *	Input	The system-wide usable carrier list

Table 5.26 – Channel Estimator Inputs and Outputs

The channel estimation algorithm is

1. Regenerate the preamble (gives the “golden reference”)
 - a. For each carrier
 - i. Copy the reference phase for the carrier
 - ii. Set the mapper input to “1” (SYNCP)
 - b. Pass the SYNCP through the mapper
 - c. For each carrier
 - i. Set the mapper input to “0” (SYNCM)
 - d. Pass the SYNCM through the mapper
2. Pass the received preamble through the FFT (fftblock) – this gives the received SYNCP and SYNCM symbols (rxSYNC)
3. Calculate the correction factors
 - a. Clear the correction factor array
 - b. For each full preamble symbol
 - i. If the symbol is SYNCP use the SYNCP symbol from 1, and if the symbol is SYNCM use the SYNCM symbol.
 - ii. Calculate the inverse (Inv) real (SYNC.real / ((SYNC.real * SYNC.real) + (SYNC.imag * SYNC.imag)))
 - iii. Calculate the inverse (Inv) imaginary (SYNC.imag / ((SYNC.real * SYNC.real) + (SYNC.imag * SYNC.imag)))

- iv. Calculate the real correction factor and add to current total
 $(rxSYNC.re * Inv.re) - (rxSYNC.im * Inv.im)$
- v. Calculate the imaginary correction factor and add to current total
 $(rxSYNC.im * Inv.re) - (rxSYNC.re * Inv.im)$
- c. Calculate the actual correction factors – for each carrier
 - i. Calculate the average real and imaginary parts
 - ii. Calculate the inverse – gives the filter coefficients
 - iii. Store the results

5.3.6 OFDM DECODER (OFDM_DECODE)

Once the channel filter coefficients have been calculated, the data (either Frame Control or Payload) is passed through the OFDM decoder. This is the inverse of the encoder, and consists of four stages which are given below, and shown in Figure 5.26.

1. Remove Cyclic Prefix (de_cycl)
2. FFT (fftblock)
3. Channel Filter (chan_filt)
4. De-Mapper (demap)

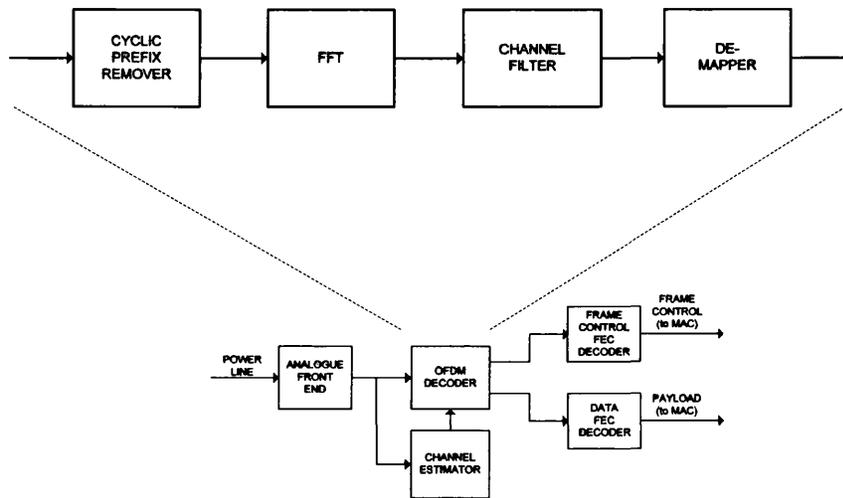


Figure 5.26 – OFDM Decoder

The inputs and outputs of the OFDM decoder are given in Table 5.27.

Variable Name	Type	Direction	Description
output	UINT8 *	Output	The decoded output binary data stream
symbols	float *	Input	The received data stream
tonemask	UINT8 *	Input	System wide Tone Mask
tonemap	UINT8 *	Input	Usable carrier list
modulation	modl	Input	Modulation used to encode data
phase	float *	In/Out	Reference phase
num sym	int	Input	Number of symbols in message (symbols)
num car	int	Input	Number of usable carriers
dir	char *	Input	The directory to store the intermediate results

Table 5.27 – OFDM Decoder Inputs and Outputs

The algorithm used in the function is

1. While there are symbols left
 - a. Determine the size of the block (40, 20 or 4 symbols)
 - b. Copy the samples that make up the block

- c. Calculate the number of symbols left (symbols – block size) and the number of bits in the output (block size * number of carriers)
 - d. Remove the cyclic prefix (`de_cycl`)
 - e. Save if requested
 - f. Pass the data through the FFT (`fftblock`)
 - g. Save if requested
 - h. Filter the data from the FFT with the channel filter (`chan_filt`)
 - i. Save if requested
 - j. Pass the filtered data through the demapper
 - k. Save if requested
 - l. Copy decoded bits to the output
2. Return the length of the output

Note that the “Save if Requested” steps are again the same algorithm as that described previously.

5.3.6.1 Remove Cyclic Prefix (`de_cycl`)

The first block in the OFDM decoder is the Cyclic Prefix remover. This removes the prefix added previously, and it also uses an offset to select data from the prefix that might have been altered during transmission (from inter-symbol interference for example). This is shown in Figure 5.27.

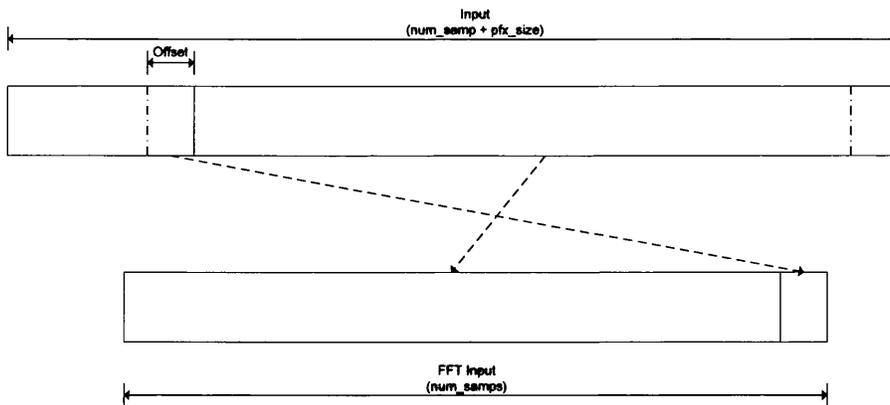


Figure 5.27 – Cyclic Prefix Remover

The inputs and outputs for the function are given in Table 5.28.

Variable Name	Type	Direction	Description
output	float *	Output	The Output data stream
data in	float *	Input	The Input data stream
num_sym	int	Input	The number of symbols
num_samp	int	Input	The number of samples in each (output) symbol

px_size	int	Input	The size of the prefix that was added
offset	int	Input	The number of samples to take from the prefix

Table 5.28 – Cyclic Prefix Remover Inputs and Outputs

The algorithm used in the function is

1. For each symbol
 - a. For each sample
 - i. If the sample is in the “offset” section, place at the end of the output
 - ii. If the sample is in the “main” section, place at the start of the output

5.3.6.2 Fast Fourier Transform (**fftblock**)

The FFT Block takes the 256 samples per symbol that is the output from the cyclic prefix removal stage and performs a Fast Fourier Transform on them. This gives the phase information that is used to convert the data back into binary. The function only returns the phase data of the 84 carriers that are used by HomePlug.

The inputs and outputs for the function are given in Table 5.29.

Variable Name	Type	Direction	Description
output	complex *	Output	The decoded output stream (contains HomePlug frequencies)
timedom	float *	Input	The input data stream
sym	int	Input	Number of symbols in the input

Table 5.29 – FFT Block Inputs and Outputs

The algorithm used in the function is

1. For each symbol
 - a. Create the FFT input signal (copy the next symbols worth of samples to the input)
 - b. Calculate the FFT of the symbol (**fft**)
 - c. Copy the data from the 84 sub-carriers used by HomePlug to the output

The FFT function used is the same as is used in the IFFT of the encoder.

5.3.6.3 Channel Filter (**chan_filt**)

The channel filter is a simple 1-tap filter that operates on each of the sub-carriers of the output from the FFT. The coefficients are those calculated by the channel estimation

function and the filter function itself is an implementation of the circuit shown in Figure 5.28.

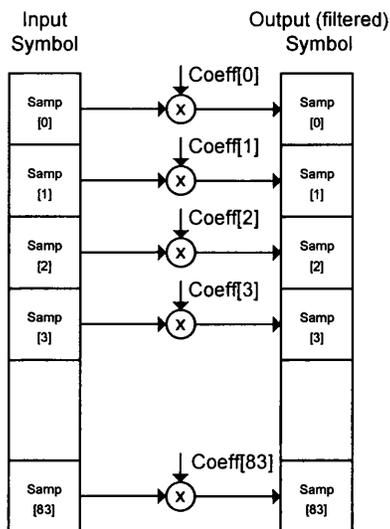


Figure 5.28 – Channel Filter Circuit

Note that the symbols and multipliers are complex.

The inputs and outputs for the function are given in Table 5.30.

Variable Name	Type	Direction	Description
fft_out	complex *	In/Out	Input data stream (from FFT) to be filtered
chan_coef	complex *	Input	Channel filter coefficients
num_sym	int	Input	Number of symbols
num_car	int	Input	Number of carriers

Table 5.30 – Channel Filter Inputs and Outputs

The algorithm used in the function is

1. For each symbol
 - a. For each sub-carrier (the 84 of HomePlug)
 - i. Get the correct input complex components
 - ii. Get the filter coefficient for the sub-carrier
 - iii. Multiply the input with the filter coefficient (complex)
 - iv. Copy the results to the output

5.3.6.4 De-Mapper (demap)

The final stage in the OFDM decoder is the de-mapper. The block performs the inverse of the mapper and converts the phase information back to a binary representation. It

does this by removing the reference phase (in the same way as it was added in the encoding process) and then determining what the most likely input was given the phase. Figure 5.29 shows the phase ranges and how these are mapped back to binary.

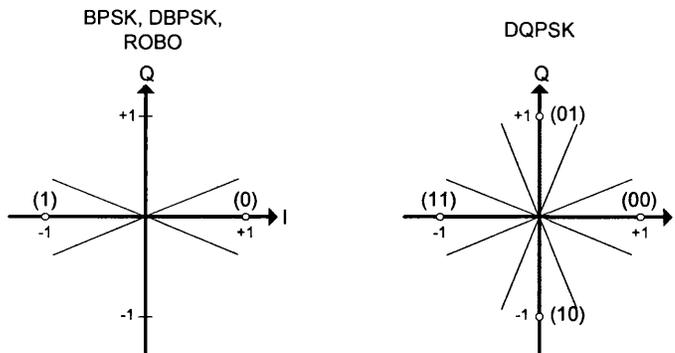


Figure 5.29 – De-Mapping Operation

The inputs and outputs for the function are given in Table 5.31.

Variable Name	Type	Direction	Description
output	UINT8 *	Output	Output binary data stream
data_in	complex *	Input	Input complex data stream
tonemap	UINT8 *	Input	Tone Map – Link specific usable carriers
tonemask	UNIT8 *	Input	Tone Mask – System wide usable carriers
mod	modl	Input	Modulation scheme used
ref	float *	In/Out	Reference phase (is updated)
num_sym	int	Input	Number of symbols

Table 5.31 – De-Mapper Inputs and Outputs

The algorithm used in the function is

1. For each symbol
 - a. For each sub-carrier
 - i. Convert the input real and imaginary parts to polar form (gives the phase)
 - ii. Remove the reference phase
 - iii. Update the reference phase (if the message is payload or the last symbol of Frame Control)
 - iv. Convert the phase to binary
 - a. BPSK, DBPSK, ROBO: If the phase is within $\pm 5\%$ of π from 0π , 2π or -2π then the binary value is 0
 - b. BPSK, DBPSK, ROBO: If the phase is within $\pm 5\%$ of π from π or $-\pi$ then the binary value is 1
 - c. DQPSK: If the phase is within $\pm 5\%$ of π from 0π , 2π or -2π then the binary value is 00

- d. DQPSK: If the phase is within $\pm 5\%$ of π from π or $-\pi$ then the binary value is 11
- e. DQPSK: If the phase is within $\pm 5\%$ of π from $\pi/2$ or $-3\pi/2$ then the binary value is 01
- f. DQPSK: If the phase is within $\pm 5\%$ of π from $3\pi/2$ or $-\pi/2$ then the binary value is 10
- v. Copy the binary value to the output, if the carrier is not blocked.

5.3.7 FRAME CONTROL DECODER (FC_DEC)

The Frame Control Decoder takes the bit stream from the OFDM decoder (which will be spread over the 4 Frame Control OFDM Symbols) and recreates the 25 bits of the Frame Control for the MAC. It consists of three functions (which are the inverse of operations of the functions in the encoder), which are given below and shown in Figure 5.30.

1. Bit Generator (`fc_bitgen`)
2. De-Interleaver (`fc_deinter`)
3. Product Decoder (`fc_deprod`)

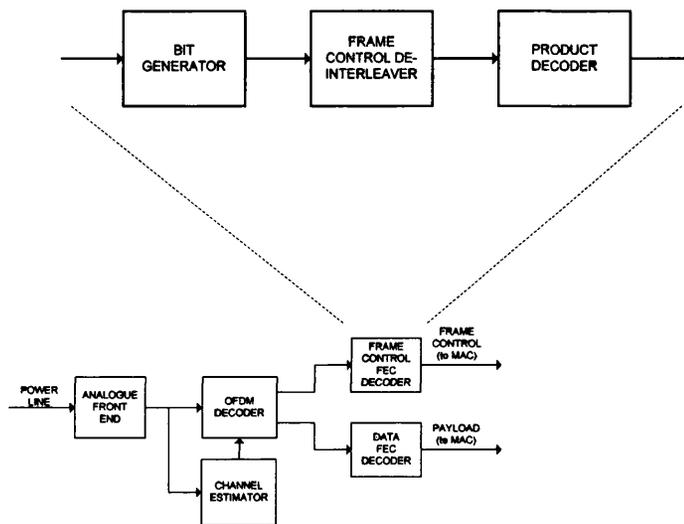


Figure 5.30 – Frame Control FEC Decoder Block Diagram

The inputs and outputs for the function are given in Table 5.32.

Variable Name	Type	Direction	Description
output	UINT8 *	Output	The decoded 25 Frame Control Bits
data	UINT8 *	Input	The input data stream from the OFDM Decoder
num_car	int	Input	The number of carriers

Table 5.32 – Frame Control Decoder Inputs and Outputs

The algorithm used in the function is simple, as it just passes the data through each stage.

1. Pass the data through the bit generator (`fc_bitgen`)
2. Pass the 100 bits from the bit generator through the de-interleaver (`fc_deinter`)
3. Pass the de-interleaved data through the product decoder (`fc_deprod`)

5.3.7.1 Bit Generator (**bit_gen**)

The bit generator takes the data from the OFDM decoder, which is spread over 4 OFDM symbols, and recreates the 100 bits for the de-interleaver. It uses a simple averaging scheme to determine the input value.

The inputs and outputs for the function are given in Table 5.33.

Variable Name	Type	Direction	Description
output	UINT8 *	Output	The 100 “bits” output
input	UINT8 *	Input	The input data stream
num_car	int	Input	The number of carriers

Table 5.33 – Bit Generator Inputs and Outputs

The algorithm used in the function is

1. Clear the arrays used in the function
2. For each of the 4 OFDM symbols
 - a. For each of the 84 carriers
 - i. Calculate the input bit position
 - ii. Calculate the output bit position
 - iii. Add the value at the input position to the value at the output position, and increment the number of values in the output
3. For each of the 100 “bits”
 - a. If the average value (output/output count) is greater than the threshold
 - i. The output is 1
 - b. Else
 - i. The output is 0

5.3.7.2 Frame Control De-Interleaver (**fc_deinter**)

The de-interleaver uses the same algorithm as the interleaver to place the bits back in their original positions ready for the product decoder stage. The only difference is the indexes used are swapped. The interleaver has the line

```
output[oldIx] = data[newIx];
```

Where as the de-interleaver has the line

```
output[newIx] = data[oldIx];
```

The inputs and outputs for the function are given in Table 5.34.

Variable Name	Type	Direction	Description
output	UINT8 *	Output	The output data stream
data	UINT8 *	Input	The input data stream

Table 5.34 – De-Interleaver Inputs and Outputs

5.3.7.3 Frame Control Product Decoder (fc_deprod)

The Product Decoder takes the 100 bits from the de-interleaver and returns the 25 bits of the input array. It will also correct errors in the input bit stream by using a hamming-based correction function which will correct single bit errors. The function first corrects the columns and then the rows, as shown in Figure 5.31.

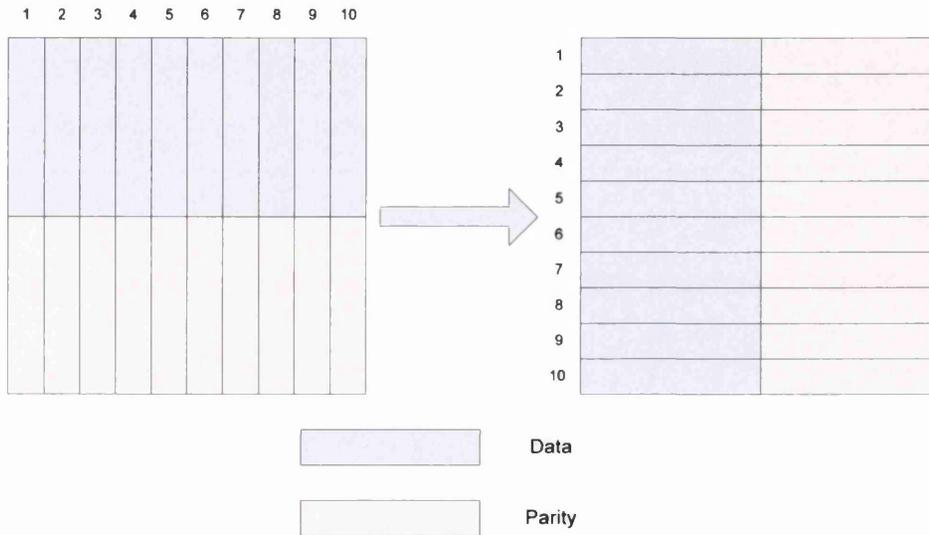


Figure 5.31 – Product Decoder Operation

The inputs and outputs for the function are given in Table 5.35.

Variable Name	Type	Direction	Description
output	UINT8 *	Output	The 25 Frame Control bits output
data	UINT8 *	Input	The input data stream

Table 5.35 – Product Decoder Inputs and Outputs

The algorithm used in the function is

1. Correct the columns. For each column in the input array
 - a. Copy the 10 bits in the column to the hamming correction function input
 - b. Use the hamming correction function to correct the errors (de_ham)
 - c. Copy the decoded data to the intermediate array
2. Correct the rows. For each row in the intermediate array
 - a. Copy the 10 bits in the row to the hamming correction function input
 - b. Use the hamming correction function to correct the errors (de_ham)

- c. Copy the decoded data to the output array
3. Copy the 25 bits of Frame Control Data to the output

The function uses the `de_ham` function to correct single bit errors in each 10-bit column and row. The algorithm used is

1. Calculate the “local” parity based on the 5 “data” bits of the input (using the same parity calculator as the encoder)
2. Calculate the syndrome (local parity x-or'd with input parity), and convert it to a decimal representation.
3. If the syndrome is non-zero (implies there is an error)
 - a. Determine the error position from the syndrome value (via a look-up table)
 - b. Correct the error

5.3.8 PAYLOAD DECODER (PAYLOAD_DEC)

The Payload Decoder re-creates the payload portion of the transmitted data and corrects any errors that might have occurred, via the Viterbi and Reed-Solomon Decoders. The function consists of five stages, which are given below and shown in Figure 5.32.

1. De-Interleaver (deinter)
2. De-Puncture (depunct)
3. Viterbi Decoder (viterbi)
4. Reed-Solomon Decoder (rs_dec)
5. De-Scrambler (descramble)

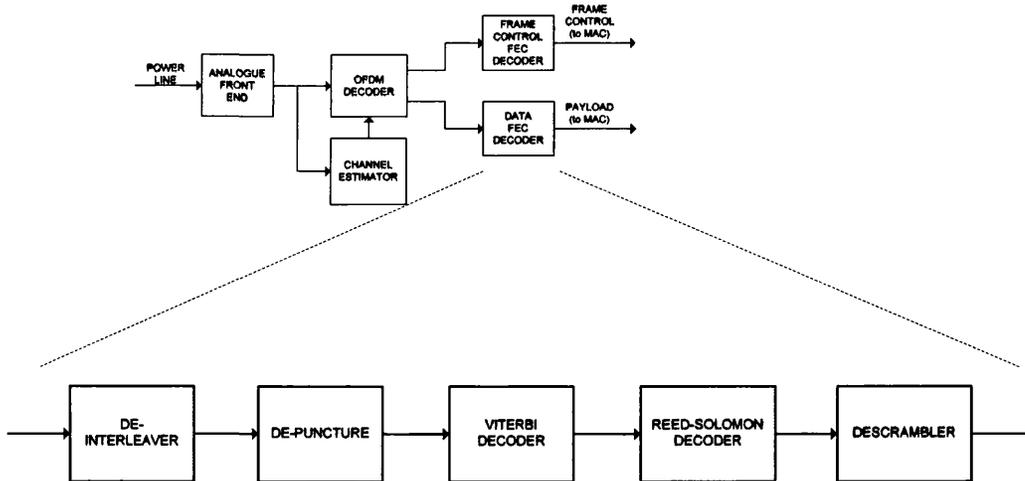


Figure 5.32 – Payload FEC Decoder Block Diagram

The inputs and outputs for the function are given in Table 5.36.

Variable Name	Type	Direction	Description
dec_rx	UINT8 *	Output	Decoded version of rx_data
rx_data	UINT8 *	Input	Received data stream
mod	modl	Input	Modulation used
mode	punct	Input	Puncturing mode
length	int	Input	Length of input data stream
num_car	int	Input	Number of carriers
num_40	int	Input	Number of 40 Symbol PHY blocks
num_20	int	Input	Number of 20 Symbol PHY blocks
dir	char *	Input	Results directory

Table 5.36 – Payload Decoder Inputs and Outputs

The algorithm (as with the encoder) isn't as simple as the Frame Control as the functions are designed to operate on PHY blocks, rather than the whole data stream. The algorithm is given below.

1. Calculate the number of bits in a 20- and 40-symbol PHY block, plus the size and number of RS blocks (using the `block_bits` function)
2. Determine the number of Reed-Solomon parity symbols for the modulation scheme (16 for DxPSK, 8 for ROBO)
3. Calculate the number of bits into the de-interleaver (number of carriers x block size)
4. Calculate the number of bits out of the de-interleaver (same as the number of bits out of the convolutional encoder in the encoding process)
5. If the code rate is three-quarters, update the number of bits out of the de-interleaver
6. For each 40-Symbol PHY block
 - a. Copy the number of bits required for the de-interleaver from the input stream
 - b. Save if requested
 - c. De-interleave the PHY block (`deinter`)
 - d. Save if requested
 - e. If the code rate is half
 - i. Decode the de-interleaved data using the Viterbi decoder (`viterbi`)
 - ii. Save if requested
 - f. else
 - i. "Insert" the missing punctured bits into the de-interleaved data (`depunct`)
 - ii. Save if requested
 - iii. Decode the de-punctured data using the Viterbi decoder (`viterbi`)
 - iv. Save if requested
 - g. If there is only one Reed-Solomon block
 - i. Decode the Viterbi output data using the Reed-Solomon decoder (`rs_dec`)
 - ii. Copy the decoded data (minus the parity data) to the de-scrambler input
 - h. else
 - i. Calculate the length of the output from the Reed-Solomon decode stage (length of Viterbi output – (number of RS blocks x number of parity symbols x 8))
 - ii. For each Reed-Solomon block
 1. Calculate the number of bits into the decoder
 2. Copy the bits from the Viterbi decoded data to the decoder input

3. Decode the data using the Reed-Solomon Decoder
(rs_dec)
4. Copy the Reed-Solomon output to the de-scrambler input
 - i. Save if requested
 - j. De-scramble the data (descramble)
 - k. Save if requested
 - l. Copy the fully decoded block to the function output
7. If there is a 20-Symbol PHY block
 - a. Copy the number of bits required for the de-interleaver from the input stream
 - b. Save if requested
 - c. De-interleave the PHY block (deinter)
 - d. Save if requested
 - e. If the code rate is half
 - i. Decode the de-interleaved data using the Viterbi decoder (viterbi)
 - ii. Save if requested
 - f. else
 - i. "Insert" the missing punctured bits into the de-interleaved data (depunct)
 - ii. Save if requested
 - iii. Decode the de-punctured data using the Viterbi decoder (viterbi)
 - iv. Save if requested
 - g. If there is only one Reed-Solomon block
 - i. Decode the Viterbi output data using the Reed-Solomon decoder (rs_dec)
 - ii. Copy the decoded data (minus the parity data) to the de-scrambler input
 - h. else
 - i. Calculate the length of the output from the Reed-Solomon decode stage (length of Viterbi output – (number of RS blocks x number of parity symbols x 8))
 - ii. For each Reed-Solomon block
 1. Calculate the number of bits into the decoder
 2. Copy the bits from the Viterbi decoded data to the decoder input
 3. Decode the data using the Reed-Solomon Decoder (rs_dec)
 4. Copy the Reed-Solomon output to the de-scrambler input
 - i. Save if requested
 - j. De-scramble the data (descramble)
 - k. Save if requested

- i. Copy the fully decoded block to the function output
8. Return the length (in bits) of the decoded data

Note that the “Save if requested” steps use the same algorithm as the encoder to determine where to save the results from the steps in the algorithm.

5.3.8.1 Payload De-Interleaver (**deinter**)

The de-interleaver inverts the effect of the interleaver and ensures the logically adjacent bits are adjacent again for the rest of the decoding process. It also uses a simple average scheme with the ROBO data (where each bit is transmitted 4 times) to determine what the correct input bit was.

The inputs and outputs for the function are given in Table 5.37.

Variable Name	Type	Direction	Description
data	UINT8 *	In/Out	The data stream to de-interleave
mod	mod1	Input	The modulation scheme
len	int	Input	The length of the data
num_car	int	Input	The number of usable carriers
block_size	int	Inptu	The size of the block (20 or 40 symbols)

Table 5.37 – Payload De-Interleaver Inputs and Outputs

The algorithm used in the function is given below.

1. Set the parameters for the de-interleaver depending on the modulation and calculate the output length
 - a. ROBO – number of rows is the number of carriers, number of columns is 10, number of symbols is 4 and the output length is the input length/4
 - b. DBPSK – number of rows is twice the number of carriers, number of columns is half the block size, number of symbols is 1 and the output length is the input length
 - c. DQPSK – number of rows is twice the number of carriers, number of columns is half the block size, number of symbols is 1 and the output length is half the input length
2. For each symbol (there is only more than one with ROBO modulation)
 - a. Calculate the starting row
 - b. For each column (as determined in step 1)
 - i. For each row (as determine in step 1)
 1. Calculate the row index ((current row + starting row) mod (number of rows))

2. If this is the first (or only) symbol, copy the current input data to the correct location in the interleaver matrix ([row index][current column])
 3. If this is the last symbol (in ROBO), add the current input data to the correct location in the interleaver matrix and check if the average value is above the threshold (determines if the binary value is 1 or 0)
 4. If this is any other ROBO symbol, add the current input data to the correct location in the interleaver matrix.
3. Read the data out of the interleaver matrix and create the data for the rest of the decode process. For each row
- a. Set the initial row index (current row)
 - b. For each column
 - i. If the modulation is DQPSK
 1. Copy the two bits from the interleaver matrix to the output (current output position and current position + length)
 - ii. If the modulation is DBPSK or ROBO
 1. Copy the bit from the interleaver matrix to the current output position in the output
 - iii. Increment the current output position
 - iv. Decrement 8 from the row index, and wrap round to the end if this is less than zero

5.3.8.2 De-Puncture (depunct)

The de-puncture block inserts a dummy value into the positions that were removed from the transmitted data. This makes the three-quarter rate back to half rate and in a form suitable for decoding via the Viterbi decoder. The operation is shown in Figure 5.33.

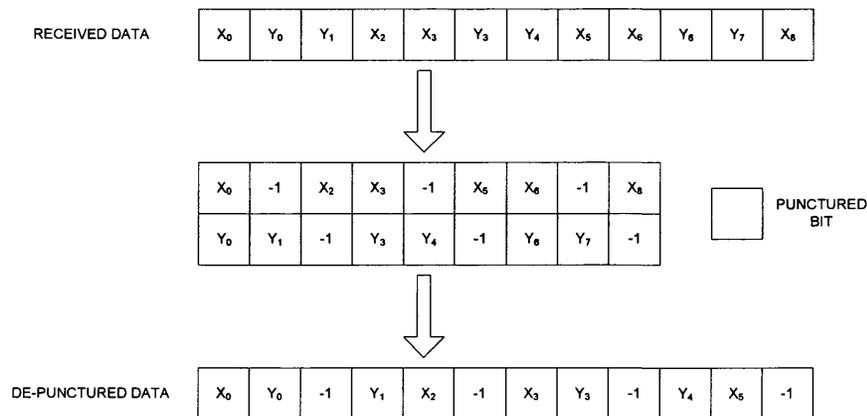


Figure 5.33 – De-Puncture Operation

The inputs and outputs for the function are given in Table 5.38.

Variable Name	Type	Direction	Description
output	UINT8 *	Output	The de-punctured output data stream
input	UINT8 *	Input	The input data stream
mode	punct	Input	The puncturing mode (half or three-quarter rate)
len_in	int	Input	The length of the input data

Table 5.38 – De-Puncturer Inputs and Outputs

The algorithm used in the function is given below.

1. De-puncture the data depending on code rate
 - a. Half rate – set return value to 0 (don't need to puncture)
 - b. Three-quarter rate
 - i. For each input bit
 1. If the position is one that was punctured, insert "-1"
 2. Copy the input bit to the output
 - ii. Set the return value to the length of the output
 - c. Any other rate set the return value to -1
2. Return the return value.

5.3.8.3 Viterbi Decoder (**viterbi**)

The Viterbi decoder is an implementation of the Viterbi Algorithm used to decode convolutional codes [78]. The Viterbi algorithm is a maximum likelihood decoding algorithm, in that it will determine the most likely path that was taken to encode the data. In this implementation it uses a hard decision process (rather than the potentially more accurate soft decision).

The inputs and outputs for the function are given in Table 5.39.

Variable Name	Type	Direction	Description
op_data	UINT8 *	Output	The output data stream
input	UINT8 *	Input	The input data stream
gen1	UINT8	Input	First generator polynomial (in decimal)
gen2	UINT8	Input	Second generator polynomial (in decimal)
K	int	Input	"K" value of encoder (number of storage elements)
tb_depth	int	Input	Trace back depth (multiplication factor)
num_data	int	Input	Amount of data in the input

Table 5.39 – Viterbi Decoder Inputs and Outputs

The algorithm used in the function is given below.

1. Calculate the actual trace back depth ("K" x "tb_depth")
2. Initialise the tables. For each state of the encoder

- a. If this is state 0, set the trellis for this state to 0 (ensures traceback works first time through)
 - b. Initialise the error metric table for this state (current error = 0, next error = 0xFF)
 - c. Set the input table to "-1" (unused)
 - d. For each possible input
 - i. Calculate the output for the X generator for this state
 - ii. Calculate the output for the Y generator for this state
 - iii. Store the output in the output table (at [state][input])
 - iv. Calculate the next state based on the current state and input, and sort in the next state table (at [state][input])
 - v. Store the input which caused the transition from state to next state in the input table (at [state][next_state[state][input]])
3. Create the hamming distance table (fixed values)
 4. Decode the input stream. For each pair of input values (X and Y)
 - a. Combine the pair into a single value. This is done in such a way that the punctured data (which is "-1" in the input stream) will act in the same way as the non-punctured data and will give a positive index into the hamming distance table
 - b. Determine the step size. If the trellis isn't full, then only certain states need considered
 - c. For each state in the trellis (increment by the step size from b)
 - i. For each input value
 1. Get the next state and output from the tables
 2. Calculate the hamming distance between the actual input and the output
 3. Update the error metric (current error + hamming distance)
 4. If the error metric is less than the next state error metric, update the next state error metric and set the trellis transition to the current state
 - d. Update the error metric table (set current to next, next to current and set each value in the next error to 0xFF)
 - e. If the trace back depth has been reached (ie the trellis is full)
 - i. Find the state with the smallest error (in the error metric table)
 - ii. Set the trellis read pointer equal to the trellis write pointer
 - iii. Trace back through the trellis. For each stage in the trellis
 1. Store the state that got us here (in the route table)
 2. Update the state from the trellis (find the path that goes to the previous stage)
 3. Decrement the trellis read pointer

- iv. Add the encoder input which would cause the transition from route[0] to route[1] to the output data stream (this information comes from the input table)
- v. Increment the trellis write pointer, looping round if needed
- 5. Flush the remaining states from the route table. For each entry in the table
 - i. Add the encoder input which would cause the transition from the current route to the next route to the output data stream (this information comes from the input table)
- 6. Return the length of the output, minus the tail bits

5.3.8.4 Reed-Solomon Decoder (`rs_dec`)

The Reed-Solomon Decoder is an implementation of the theory given in [78]. The decoder consists of five sub-functions, given below and shown in Figure 5.34.

1. Syndrome Calculator (`syn_calc`)
2. Berlekamp-Massey Function (`berlmas`)
3. Chien Search (`chien`)
4. Omega Function Generator (`omega_gen`)
5. Error Magnitude Calculator (`error_mag`)

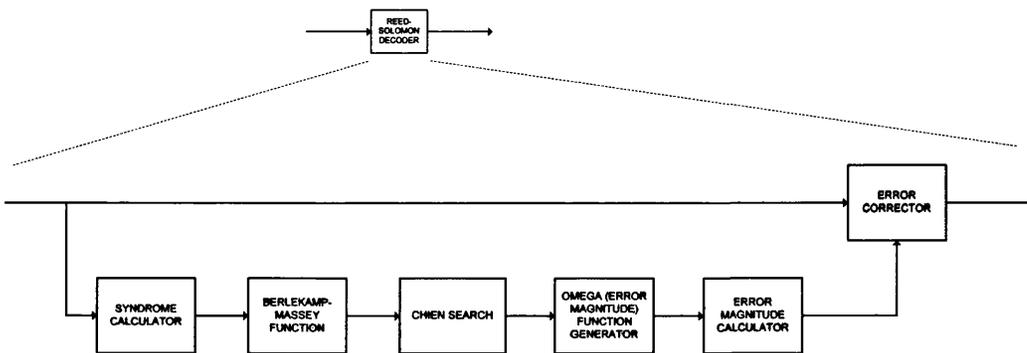


Figure 5.34 – Reed Solomon Decoder Block Diagram

The inputs and outputs for the function are given in Table 5.40.

Variable Name	Type	Direction	Description
msg	UINT8 *	In/Out	Received message (in binary). At end of function will contain the corrected version
length	int	Input	The length of the data (in bits)
modulation	mod1	Input	The Modulation scheme used

Table 5.40 – Reed-Solomon Decoder Inputs and Outputs

The algorithm for the top-level function is given below

1. Set the number of parity symbols based on the modulation (8 in ROBO, 16 in DxPSK)
2. Create the conversion arrays (*fieldgen*)
3. Convert the binary input to Reed-Solomon symbols in Power representation (8 bits per symbol, using the array from step 2)
4. Calculate the syndromes (*syn_calc*)
5. If any syndromes are non-zero
 - a. Use the Berlekamp-Massey function to find the Error Locator Polynomial, *lambda* (*berlmas*)
 - b. Find the roots of *lambda* and their positions (*chein*)
 - c. If there were more than 0 roots then correct them
 - i. Shift the syndromes by one
 - ii. Calculate the error magnitude polynomial (*omega_gen*)
 - iii. Calculate the error magnitudes (*error_mag*)
6. Correct the errors (if detected). For each error root
 - a. Calculate the actual position of the data in the input array
 - b. Add (using Reed-Solomon add) the input in the position to the error magnitude value for this root
7. Convert the decoded message back to binary

The algorithm used in the syndrome calculator is based on the following equation, and is given below

1. Clear the syndrome array
2. For each Reed-Solomon Symbol in the input
 - a. For each parity symbol
 - i. Calculate the value of the alpha power
 - ii. Calculate the partial summation (current input symbol + alpha power)
 - iii. Add the partial summation to the syndrome
3. Check how many non-zero syndromes there are. For each syndrome
 - a. If it is not zero, increment the non-zero count
4. Return the non-zero count

The Berlekamp-Massey function implements the algorithm given in [78]

The Chien Search searches the entire Reed-Solomon field to determine the roots of the ELP. The algorithm is

1. For each possible value in the Reed-Solomon field (i)

- a. Calculate the value of lambda at i
- b. If the answer is zero
 - i. Copy the value into the root array
 - ii. Determine the position (field size – i)
 - iii. Increment the number of roots
2. Return the number of roots

The `omega_gen` function creates the error magnitude polynomial, from which the amount that should be added to the input to correct it can be determined. The algorithm is given below.

1. For each syndrome, i
 - a. Reset the summation variable
 - b. Calculate the coefficient for location i
 - c. Store the variable in omega

The final function “solves” the error magnitude polynomial and gives the actual value used to correct the error. The algorithm is given below.

1. For each error
 - a. Calculate the value of the inverse of alpha
 - b. Evaluate the numerator
 - c. Evaluate the denominator
 - d. Calculate the error magnitude (numerator/denominator)

5.3.8.5 Descrambler (**descramble**)

The de-scrambler is exactly the same as the scrambler. It exploits the fact that doing the xor again (with the same value from the pseudo-random sequence) will give you the original value. After this stage the output data should be the same as the input (unless there were too many errors to correct) and is ready to be passed back to the MAC.

The inputs and outputs of the function are given in Table 5.41.

Variable Name	Type	Direction	Description
data	UINT8 *	In/Out	The data stream to be scrambled
length	int	Input	The length of the data (in bits)

Table 5.41 – Scrambler Function Inputs and Outputs

The algorithm of the function is

1. Initialise the pseudo random sequence to all ones

2. For each bit in the data
 - a. Calculate the exclusive-or value ($x^7 \oplus x^4$)
 - b. Shift the sequence to the left (ie $x^7 = x^6$, etc)
 - c. Store the exclusive-or value in x^1
 - d. Set the output value equal to the input value exclusive-or'd with the input value

5.4 CHANNEL MODEL

The channel model is used to mimic the effects of the physical channel that the data signals travel down. In this version of the networking model this is the power line. The characteristics of the power line for data communication are described in Section 3.4. These effects are modelled using a filter. There are two aspects to the channel model:

- Generation of the filter coefficients
- Modelling the data transmission

The generation of the coefficients is done before the model starts. This in effect sets the layout for the model. This is the protocol specific part and determines how the data is altered as it is transmitted. The filter coefficient generator is a very simple one. The reason for this is that the work required to generate a full channel model is beyond the scope of this research, and so a slightly more simplified approach was taken. This uses the physical characteristics of the power line cables to determine the characteristic impedance and propagation constants of a length of cable. From this the impulse and frequency responses of a specified network can be calculated.

The actual channel model used within the network system uses these pre-calculated filter coefficients to alter the data as it would in a real system. There are two sets of filters for every host to destination pair; a common transmit filter and a destination specific receive filter.

The transmit filter is used to model any effect that the Analogue Front End (AFE) and the connection of the HomePlug device onto the network might have on the signal (i.e. it will model those aspects that are common to all destinations). The receive filter is used to model the effect of the data signal travelling down a particular path to a destination. This is shown in Figure 5.35.

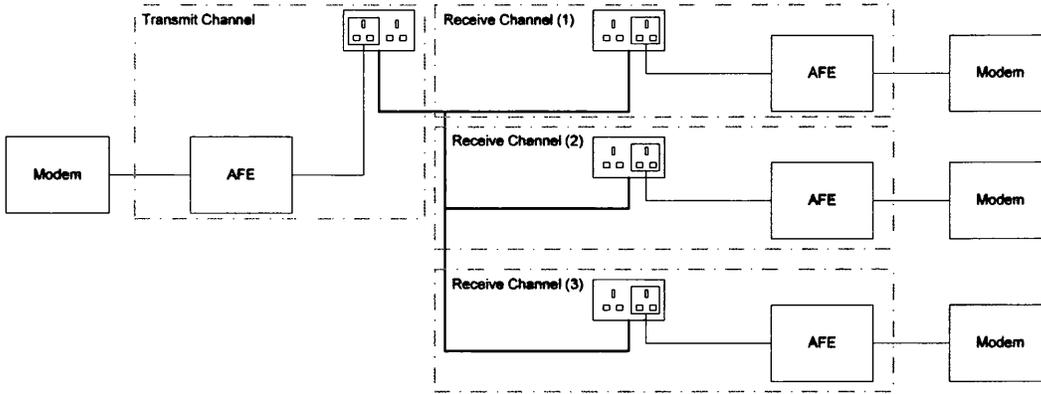


Figure 5.35 – Channel Model

When the transmission is underway, the controller can update the coefficients at the end of each OFDM symbol. This allows the model to be as dynamic as the channel would be in real life and model such things as new appliances being plugged in, changes to the topology (such as extension cables being plugged in), etc. This is possible because each receiving node acknowledges the symbol and the next symbol is not transmitted until the last node has acknowledged the current symbol.

Figure 5.36 shows the message passing between the transmitting node and the channel and receiving node (1) as well as the reloading of the channel filter coefficients during transmission (2).

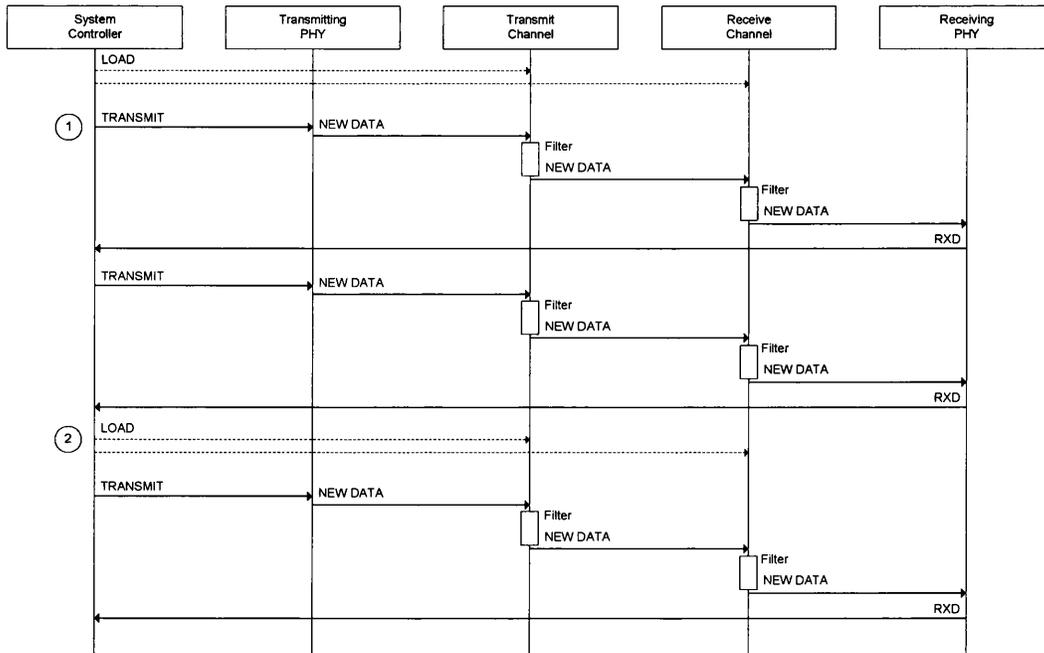


Figure 5.36 – Channel Model Message Passing

The channel components have a simplified message handling routine (compared to the Node components) as they don't have to respond to the complex message sequences that the SoC, MAC and PHY components do. The algorithm for the transmitting and receiving channels are given below.

Transmitter

1. While Running

a. Wait for Message

b. Process Message

- i. **STOP:** Set the stop flag
- ii. **NEWDATA:** If the data is a frame (i.e. not PRS) filter using the current filter coefficients, copy it to the output buffer and then send a NEWDATA event to each receive channel. If the data is a Priority Resolution symbol, determine if the transmitting station has sent a "PRS1" or a "PRS0" and wait until all the nodes have sent one. If any node has sent a PRS1 symbol send this otherwise send a PRS0.
- iii. **LOAD:** Store the filter coefficients
- iv. **START:** Clear the filter memory stages.

Receiver

1. While Running

a. Wait for Message

b. Process Message

- i. **STOP:** Set the stop flag
- ii. **NEWDATA:** Filter the data using the current filter coefficients, copy it to the output buffer and then send a NEWDATA event to the receiving PHY
- iii. **LOAD:** Store the filter coefficients
- iv. **START:** Clear the filter memory stages.

5.5 SUMMARY

In this chapter the functions that were developed to model the way in which the HomePlug model modifies the data stream was presented. The MAC and PHY functions for both encoding and decoding the data were given, and these are fully orthogonal. If data is passed through the system, assuming that there are not too many errors introduced, then the data at the output of the decoder will be identical to the data at the input of the encoder.

Chapter 6 - Results

Provides the results of using the model developed to explore some simple use cases. It also shows how the model can be used to explore alternative algorithms or implementations.

6.1 INTRODUCTION

In the previous chapters the network/hardware model that was developed was described. In this chapter the model is used to run some simple tests to prove the concept of the model and also to give a degree of validation. The validation is done by comparing the model results with the theoretical results presented in [34]. As the model is designed to allow hardware exploration, the final test looks at the effect on latency that different buffer sizes have.

The simulations run are

1. Typical home environment use case
2. Throughput verses number of nodes
3. Latency verses buffer size

Details of each test are given in the appropriate sections, along with the results. The chapter closes with a summary of the findings.

The channel model, as developed, doesn't properly model the channel (this is an area of future work) and so for the simulations run here the "NO_CHAN" option was used. However different modulation schemes, number of carriers, etc., were simulated using pre-set Tone Maps. These allowed modelling (to an extent) of various conditions. A Perl script was developed which would create "good", "average" and "bad" channel conditions. These had the characteristics given in Table 6.1

Channel Type	Number of Carriers	Modulation	Code Rate
Good	45-76	70% DQPSK 25% DBPSK 5% ROBO	80% $\frac{3}{4}$ Rate 20% $\frac{1}{2}$ Rate
Average	25-50	70% DQPSK 25% DBPSK 5% ROBO	80% $\frac{3}{4}$ Rate 20% $\frac{1}{2}$ Rate

Channel Type	Number of Carriers	Modulation	Code Rate
Bad	10-35	70% DQPSK 25% DBPSK 5% ROBO	80% $\frac{3}{4}$ Rate 20% $\frac{1}{2}$ Rate

Table 6.1 – Channel Characteristics

Although these aren't actual channels, they do allow exploration of behaviour under different conditions.

6.2 SOFTWARE SUITE

The software developed for the model described in the previous chapters is extensive, running to around 50,000 lines of C. It will operate on any Linux/GNU machine. The primary development platforms were a native Linux PC and a cygwin environment running on a Windows PC.

The model would easily port to other machine types, such as Solaris, as long as a working C compiler was available and the pthreads library present.

6.3 TYPICAL USE CASE

This test shows the model running a typical scenario of a home network. It involves 5 nodes, representing various devices throughout the home, such as a server, video display, Network Attached Storage (NAS), PC's, etc. The test setup is shown in Figure 6.1, with Table 6.2 describing what data is being transmitted.

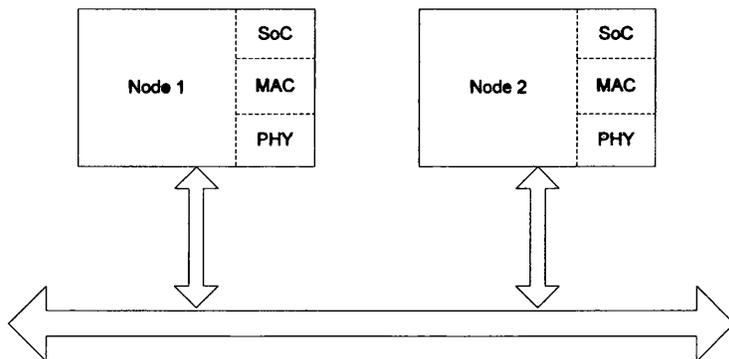


Figure 6.1 – Typical Home Network Scenario

Source	Destination	Traffic Type	Data Rate	Priority
Router	PC1	VoIP	5 kbps	High
Router	PC2	Net Traffic	200 kbps	Low
Media Server	Display	Video Stream	1 Mbps	Medium
Media Server	PC1	Audio Stream	128 kbps	Medium
PC2	NAS	Data	500 kbps	Low

Table 6.2 – Network Traffic

The scenario was run for the three channel types, with the results given in Table 6.3. An example output from the program is given in Figure 6.2, although as the simulator is text/command-line based, this isn't very exciting!

Node		MAC Results			SoC Results		
		Throughput (bps)	Bytes Received ¹	Avg. Latency (ms)	Throughput (bps)	Bytes Received	Avg. Latency (ms)
1 Router	Good	NA	0	NA	NA	0	NA
	Avg						
	Poor						
2 Media Server	Good	NA	0	NA	NA	0	NA
	Avg						
	Poor						

¹ Note this doesn't consider the response frames.

Node		MAC Results			SoC Results		
		Throughput (bps)	Bytes Received ¹	Avg. Latency (ms)	Throughput (bps)	Bytes Received	Avg. Latency (ms)
3 PC 1	Good	12568	39300	1.2	12040	37650	4.7
	Avg	12479	39300	1.8	11934	37650	6.3
	Poor	12310	39300	2.5	11845	37650	9.75
4 PC 2	Good	363240	171666	1.57	329592	155764	5
	Avg	341765	171666	2.03	307588	155764	6.29
	Poor	334157	171666	2.76	300741	155764	8.94
5 Display	Good	1153008	111852	1.36	1060008	102831	4.22
	Avg	1152018	111852	1.86	1036816	102831	5.76
	Poor	1047651	111852	2.5	987885	102831	7.75
6 NAS	Good	574208	113764	1.84	524856	103987	3.61
	Avg	532100	113764	3.1	478890	103987	9.61
	Poor	452310	113764	6	407079	103987	18.6

Table 6.3 – Home Network Scenario Results

```

NODE 0 Results
*****
MAC Bytes Received      = 0
MAC Thruput             = nan bytes per second
MAC Thruput             = 0 bits per second
MAC Average Frame Latency = nan ms
MAC Failed MSDU's      = 0

SoC Bytes Received     = 0
SoC Thruput            = nan bytes per second
SoC Thruput            = 0 bits per second
SoC Average Latency    = nan ms
Number of messages transmitted = 164
Number of messages received = 0
Number of failed messages = 0
*****

NODE 1 Results
*****
MAC Bytes Received      = 0
MAC Thruput             = nan bytes per second
MAC Thruput             = 0 bits per second
MAC Average Frame Latency = nan ms
MAC Failed MSDU's      = 0

SoC Bytes Received     = 0
SoC Thruput            = nan bytes per second
SoC Thruput            = 0 bits per second
SoC Average Latency    = nan ms
Number of messages transmitted = 69
Number of messages received = 0
Number of failed messages = 0
*****

NODE 2 Results
*****
MAC Bytes Received      = 39300
MAC Thruput             = 1571.70 bytes per second
MAC Thruput             = 12568 bits per second
MAC Average Frame Latency = 1.201200 ms
MAC Failed MSDU's      = 0

SoC Bytes Received     = 37650
SoC Thruput            = 1506.708374 bytes per second
SoC Thruput            = 12040 bits per second
SoC Average Latency    = 4.723488 ms
Number of messages transmitted = 35
Number of messages received = 25
Number of failed messages = 0
*****

NODE 3 Results
*****
MAC Bytes Received      = 171666
MAC Thruput             = 45405.51 bytes per second
MAC Thruput             = 363240 bits per second
MAC Average Frame Latency = 1.572577 ms
MAC Failed MSDU's      = 0

SoC Bytes Received     = 155764
SoC Thruput            = 41193.351562 bytes per second
SoC Thruput            = 329592 bits per second
SoC Average Latency    = 5.027520 ms
Number of messages transmitted = 0
Number of messages received = 105
Number of failed messages = 0
*****

NODE 4 Results
*****
MAC Bytes Received      = 111852
MAC Thruput             = 144126.62 bytes per second
MAC Thruput             = 1153008 bits per second
MAC Average Frame Latency = 1.365521 ms
MAC Failed MSDU's      = 0

SoC Bytes Received     = 102831
SoC Thruput            = 132501.203125 bytes per second
SoC Thruput            = 1060008 bits per second
SoC Average Latency    = 4.222887 ms
Number of messages transmitted = 0
Number of messages received = 69
Number of failed messages = 0
*****

NODE 5 Results
*****
MAC Bytes Received      = 113764
MAC Thruput             = 71776.72 bytes per second
MAC Thruput             = 574208 bits per second
MAC Average Frame Latency = 1.844357 ms
MAC Failed MSDU's      = 0

SoC Bytes Received     = 103987
SoC Thruput            = 85607.804688 bytes per second
SoC Thruput            = 524856 bits per second
SoC Average Latency    = 3.616139 ms
Number of messages transmitted = 0
Number of messages received = 69
Number of failed messages = 0
*****

```

Figure 6.2 – Home Network Scenario Simulation Output

This test shows the ease with which a simulation can be run. It took approximately 5 minutes to set up (creating the command file and tone maps), and each simulation run took about 10s.

6.4 THROUGHPUT VERSES NUMBER OF NODES

This test is designed to provide some validation of the model by comparing the results with the theoretical results presented in [34]. The test involves running the system at saturation and increasing the number of nodes. For each run, the average throughput is measured. The theory states that as the number of nodes increase, the average throughput will decrease.

The test setup is similar to that of the previous simulation, but multiple runs were carried out, each one increasing the number of nodes by one. The setup is shown in Figure 6.3.

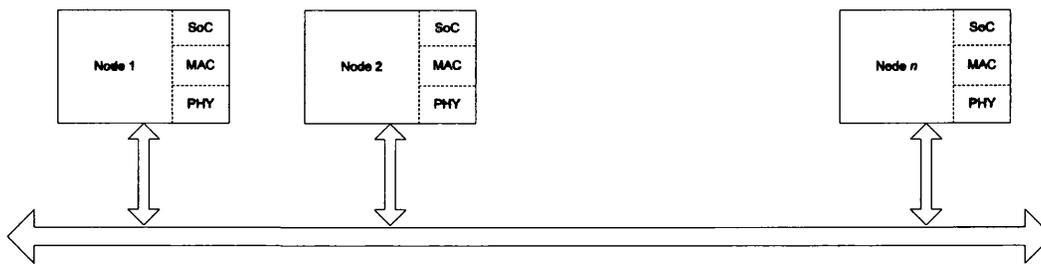


Figure 6.3 – Throughput Simulation Setup

To keep the test simple, a good channel was used throughout, and each node just sent traffic to the next node in line (i.e. Node 1 sent to Node 2, Node 2 sent to Node 3, etc.). The results of this test are summarised in Table 6.4, and shown graphically in Figure 6.4

Number of Nodes	Average Throughput		Number of Nodes	Average Throughput	
	MAC	SoC		MAC	SoC
2	6.89	6.2	11	4.96	4.51
3	6.62	6.02	12	4.82	4.43
4	6.43	5.92	13	4.76	4.29
5	6.2	5.66	14	4.63	4.17
6	5.89	5.35	15	4.5	4.09
7	5.62	5.11	16	4.47	4.05
8	5.43	4.93	17	4.43	3.99
9	5.25	4.88	18	4.36	3.96
10	5.01	4.55	19	4.32	3.93

Table 6.4 – Throughput Simulation Results

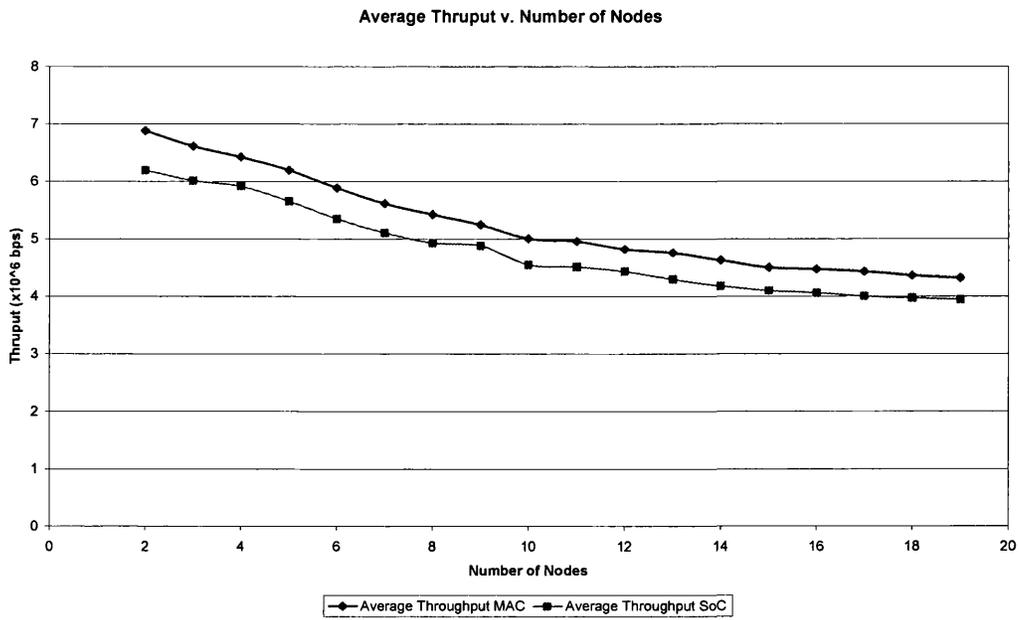


Figure 6.4 – Throughput Simulation Graph

As can be seen in Figure 6.4, as the number of nodes increase, the throughput decreases, broadly following the results presented in [34]. The discrepancies in the results follow from the fact that [34] assumes all transmissions are DQPSK with $\frac{3}{4}$ Rate encoding, whereas the model allows all modulation and code rates, along with differing numbers of channels.

6.5 LATENCY VERSES BUFFER SIZE

The final test case ran highlights one of the uses of a model such as the one presented, namely hardware exploration. In this case it is the buffer size that is being explored as this has a direct relation to the amount of RAM needed, and consequently the price of any chip developed. Equally the model could be used to explore different hardware architectures, such as logarithmic multipliers [91].

In this test the number of nodes is kept constant at 4, but the buffer size (or more accurately the number of buffers) is increased. In the test all the traffic is sent to Node 1, with the setup used shown in Figure 6.5

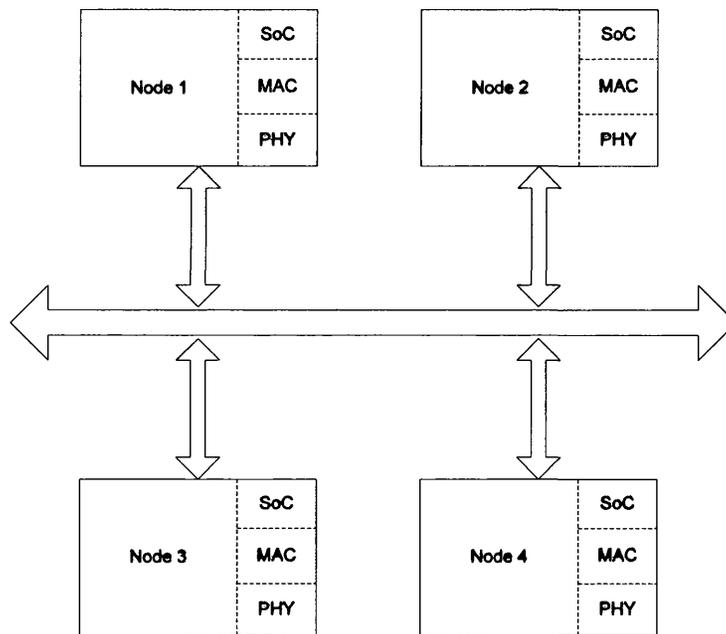


Figure 6.5 – Latency Simulation Setup

The traffic sent is given in Table 6.5, with all the traffic being the same priority

Source	Destination	Rate
2	1	1 Mbps
3	1	1 Mbps
4	1	1 Mbps

Table 6.5 – Latency Simulation Traffic

The number of receive buffers is varied from 1 to 4, and the average latency is measured. The results are shown in Table 6.6.

Number of Buffers	Average Latency (ms)
1	4.31
2	2.22
3	1.2
4	1.19

Table 6.6 – Latency Simulation Results

This test has shown how the model can be used to explore various hardware options. For example in the above scenario, if low latency was required the designers could make a decision based on the results gathered from running the test. NB The results for 3 and 4 buffers are similar as there is only 3 data sources.

6.6 SUMMARY

In this chapter the model was used to run various simple scenarios, aimed at verifying the model and exploring some of the uses such a model might have. The tests ran included:

1. Typical Use Case
2. Saturation Throughput Simulation
3. Hardware Exploration

By running these tests not only was the basic concept of the model verified, but one of the primary uses of the model was explored.

This final test is likely the most useful application of this model in an industrial use. Early decisions on the amount of RAM needed or the accuracy (or what ever criteria is relevant) of different implementations can have many benefits. These include helping to determine how costly a given implementation might be, which is an important factor to consider in the competitively priced consumer electronics market.

Chapter 7 - Conclusions

Concludes the work presented previously, by giving a summary of the problem area and then the model that was developed. It finishes with a discussion of future work that could be carried out.

7.1 INTRODUCTION

This chapter concludes the work, and provides a summary of the research that has been carried out and the reasons behind it. It also introduces areas of further work that could be carried out based on the hardware/networking model developed.

The chapter is structured as follows:

1. A re-statement of the original problem, and why it is a problem.
2. The solution developed to solve the problem.
3. Future work.

7.2 THE PROBLEM

In this section the problem that the work solves is re-iterated, along with *why* it is a problem. The problem that was solved here was how to model two aspects of the next generation of home network, namely the network itself and the hardware it is “implemented” on. This is important to do for a variety of reasons.

First of all consumer electronics have an increasingly small time-to-market. This has a massive impact on any products developed as the general rule of thumb has been that the company that gets a product to the market first wins the lion’s share of the market. Examples of this are Apple with their iPod, or Sony with the Playstation. From this, any tool or solution which helps a company get their product to market quicker will be beneficial. It is in this area that modelling is important as it gives the company many advantages in reducing the time-to-market. This can include reducing the time it takes for the design engineers to understand the product by getting a better understanding of the technical issues, and reducing the time it takes to explore alternative solutions well before a final solution is committed to.

Consumer electronics are also very cost-sensitive, and having a realistic model before the hardware is designed can ensure that the engineers have a better level of confidence in the final hardware. This is necessary to reduce the change of having to re-spin the silicon for an ASIC for example, as these can cost \$1 million or more a time with today’s technology. This is also tied into the time-to-market issue, as the time required to produced the initial silicon, detect any bugs, solve the bugs and re-produce the silicon is quite large, and will delay the company getting a product onto the market quickly.

A second reason that producing a model of the hardware and network is important is it allows engineers to explore how all the components within the system will interact. As the components are very likely to be System-On-Chip (SoC) devices, and therefore have limited processing capacity, knowing in advance where the bottlenecks within the system are likely to be will mean that solutions to the problem can be found quickly, for example by increasing the memory or the speed of the system. It could also work in the opposite way and highlight areas that might be over-specified and mean the company could make savings (in terms of area or power for example) in the final product.

A third reason for creating a model of this nature is to explore how it will interact with other networks in the vicinity. This is not directly relevant for the HomePlug network developed here, but would be when exploring how an 802.11 network and a Bluetooth network would interact for example, as both use the same frequency range to transmit. This is an important issue for home networks especially, as they are likely to be an amalgamation of multiple networking technologies, although the predominant (at least in the “no new wires” area) is wireless, and this is likely to remain the case.

A final reason for creating this type of network/hardware model is to enable researchers to explore new network protocols, based on the findings/simulations of current protocols. This becomes important when the different traffic patterns for a home network are considered for example. There is likely to be much more streaming media (video, audio, voice) on a home network than an office data network for example, and the protocols running on home networks need to be robust enough to offer a sufficient Quality-of-Service for this type of traffic.

These areas, when considered together, pose a new and interesting area to explore. Traditionally network and hardware models were considered in isolation, and the research in these two areas has been pretty much in isolation. As the two areas merge they then have to be considered together – one has as big an impact on over all functionality as the other. The days of networks consisting solely of large, general purpose computers/servers is past, and with the advent of System-on-Chip technology is a trend that is likely to continue, especially within the home.

7.3 THE SOLUTION

In this section the solution that was developed for the problem is described. The solution takes the form of a discrete event simulation environment for a System-on-Chip based HomePlug home network. The general structure of the model developed is shown again in Figure 7.1.

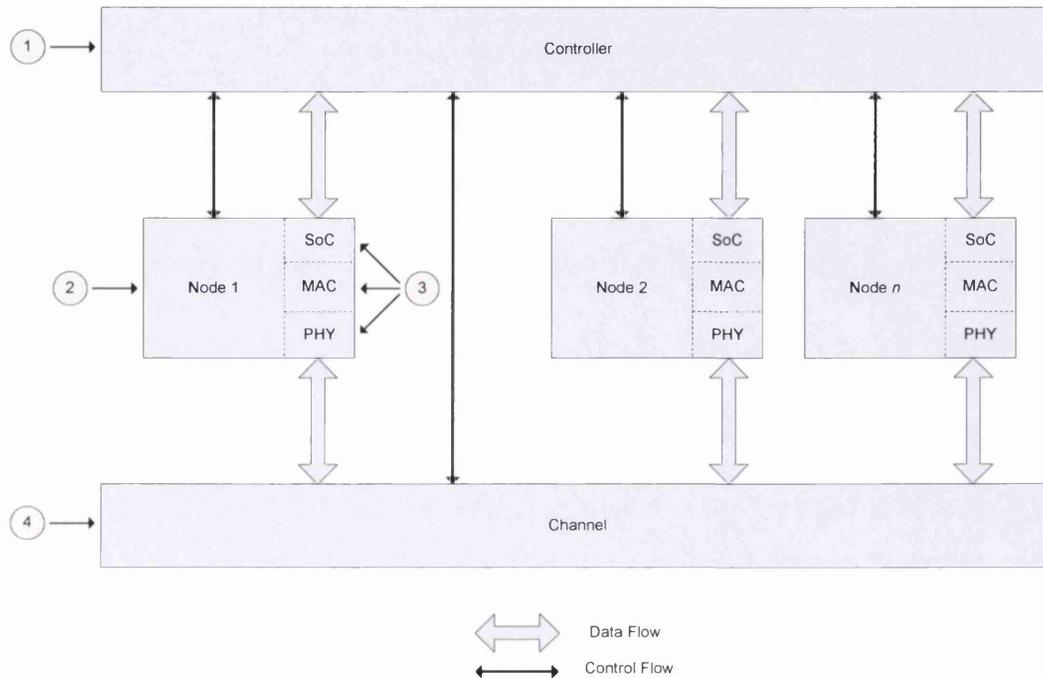


Figure 7.1 – Model Structure

The model developed consists of a controller (1) and multiple network nodes (2). The nodes communicate with each other via a common channel (4), by following the HomePlug protocol (which is a Carrier Sense, Multiple Access with Collision Avoidance scheme similar to 802.11). Internally the nodes consist of what have been termed “node-threads” (3) which communicate with each other, and the system controller, via a discrete event system.

Each node thread will block until it receives an event and will then determine whether it should process it or not (this choice is needed to accommodate events which arrive in the wrong order). This system allows the node-threads to run independently, and only stop when they need to communicate with one another. This would allow the model to run on a multiprocessor system for example. One problem that was found with the

method however was the need to ensure the event/message passing was done correctly, as if not then the entire model would block waiting on an event that never arrived.

The model developed allows the nodes to interact as they would in a real system, firstly with each other using the HomePlug protocol and secondly internally, in this case using an AMBA (Advanced Microcontroller Bus Architecture) AHB/APB (Advanced High-Speed Bus/Advanced Peripheral Bus) model, although in this case it doesn't completely follow the AMBA specification.

The model allows the exploration of various simulation runs with the aim of proving the various points raised in the previous section, and in previous chapters. Examples of the model running were given in the previous chapter and these also show another feature of the model, namely the exploration of alternative algorithms.

The model meets many of the original requirements laid out in Chapter 4, section 4.2. These are repeated here, along with a summary of why and how well the model meets them, in Table 7.1.

Ease of Use	In terms of the data manipulation aspects, the model is easy to use as it is just a C implementation of the algorithms used to modify the data. The interaction between the node threads is quite complex however, and in a future system this would benefit from some simplification.
Event Based	The model was designed from the outset to be event based, and the implementation makes extensive use of events to control every aspect of the model.
Simulation Based	This was also used as a basis of the implementation of the model, and the final solution uses a simulation approach (as opposed to an analytical or emulation approach)
Multiple Nodes	Again the model was designed from the outset to allow multiple nodes. As the nodes all follow the HomePlug channel access mechanism, there is no limit on the

	number of nodes that can run within the model, beyond the limits of the system the model is running on.
Multiple Levels	The model was designed to model more than one layer in a network protocol stack, and this it does, although the focus is on the lower layers. This focus was intentional for the initial model developed here, and one future area of work would be to extend this to the other layers of the protocol stack.
Data Metrics	The model provides basic metrics on average Throughput and frame latency. There is sufficient information in the received frames to calculate other metrics if needed.

Table 7.1 – Modelling System Mandatory Requirements

Neither of the non-mandatory requirements (Multiple Protocols and Usable with Hardware Simulators) was implemented, however these were deemed to be “nice to have” features. Even without them the model developed still solves many of the issues and points highlighted in Chapter 2 and the previous section of this chapter.

From the points above it can be seen that the model developed does do what it was intended to do. That is to model a network and the hardware running it and allow engineers to explore alternative implementations. There are, however, areas which could be expanded upon, and these are described in section 7.5.

7.4 EVALUATION OF WORK

The model developed has an important role to play in the development of System-on-Chip based networking components. As it stands it is a proof of the concept and ideas that have been presented and highlighted in previous chapters. Some of the areas which could be looked at to progress the model from a proof of concept to a useful tool are presented in the next section.

As was shown, the model can be used to explore different hardware implementations, in this case the size of buffers within the design. As mentioned a decision on this early on in the design cycle is beneficial, especially if cost is an issue. This means the designers can have an early feel of the amount of memory their system needs to meet the criteria that they have set (which will depend on the final application).

As the model is a tool, it is useful to see how it might fit in with any existing development flow. Generally SoC based solutions are designed using a flow similar to the one in Figure 7.2.

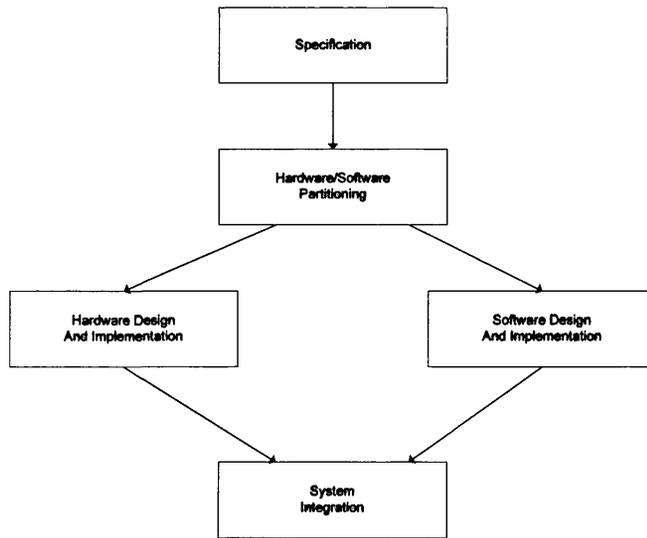


Figure 7.2 – SoC Design Flow

This, however, only shows one aspect of the system, this is how the device works in isolation, and if incorrect assumptions are made of its processing power then the device will not perform as expected. If the model presented here were used it could be used to determine if the overall requirements of the network can be met. For example, will the

solution have enough processing power to process streaming video frames? This is an important factor that is often overlooked, and the model presented here would help to address this.

7.5 FUTURE WORK

In this section some possible areas of future work for the model are highlighted. Although the model that was developed meets most of the original requirements, there are areas it would be interesting and beneficial to explore further.

An obvious first area of future work would be to expand the model for multiple network protocols. As mentioned, home networks are likely to include many different technologies, and exploring how these interact would be of great benefit. It would also allow the engineers to design the bridge between the networks and determine what the processing/memory requirements would be for this. It would also allow for an exploration of the issues of protocols that use a similar (or indeed the same) channel to transmit on, such as Bluetooth and 802.11. As the HomePlug protocol is very similar to 802.11, this would be an idea first candidate for developing the model to run with different protocols.

Following on from the point above, another area of future work would be to extend the number of network protocol levels that are implemented. Currently it is a limited Link-Layer (via the SoC), the Media Access Control (MAC) and Physical (PHY) layers that are modelled. However if the protocol stack modelled by the SoC was extended to include the Network, Transport and Protocol layers for example, a fuller picture of the network could be gained, and things like Universal Plug-n-Play (UPnP) could be modelled. UPnP is a higher level protocol that is based around XML and allows disparate devices to communicate with each other. It is being proposed as a possible solution to creating a true plug and play home network where devices from many different vendors can easily communicate with each other. By extending the model to include these aspects, product engineers would be able to understand more fully how the final product will operate, and ensure that their product will interact correctly with others using the same protocol.

The future work points raised above are fairly large extensions to the original model. Some simpler things that could be done are to develop a graphical front-end to the modelling system and to improve the channel model.

A graphical front end is almost a mandatory extension to the model as nearly every tool developed today has one. In terms of the network model, this would allow the user to very quickly describe the network that is being modelled (i.e. the topology of the network). It would also show graphically the passage of frames within the system, and allow the model to be paused to explore (or change) the network.

Another area of future work would be to develop a better power-line channel model. The one used in the system currently is based on physical characteristics of the wire, and is taken from various sources and it doesn't provide any attenuation information or a noise model. It would be very beneficial to develop a fully working model, possibly implementing it in SimuLink, and linking it into the model. However, this is an entire research topic in its own right.

A final area of work which would be useful to build on is integrating the model with standard hardware simulators. This would allow the model to be used to verify the correct operation of any hardware developed. This point was identified as being a primary use of models, and so the extension of this model to provide this function would add to its usefulness.

There are many areas that could be explored based on the work presented here. All are interesting and beneficial as the home networking market grows over the coming years, and the products that operate on them become more diverse and pervasive.

7.6 SUMMARY

In this chapter a review of the work has been carried out, detailing the problem that was originally identified, why it is a problem, and the requirements of a solution to it. The problem was how to model the hardware and networking components of a home networking system. Once the problem was introduced again, the solution that was developed was summarised, and an indication of the original requirements that it met were given.

The final section focused on potential areas of future work, and highlighted some of the more important areas. These included modelling multiple protocols, modelling higher networking layers/protocols, improving the channel model, adding a graphical interface and incorporating the model into a hardware simulator. If these areas are addressed then the model would cover all the initial requirements (both those deemed mandatory, which are solved with the current model, and those deemed non-mandatory).

The work presented here introduces a new way of looking at modelling hardware and networks, and one which will become increasingly important over the next ten years or so as the next generation of home networks take off. This is largely due to the completely different way in which these networks will operate, as no longer will they be “traditional” networks consisting of PCs connected together via Ethernet, but instead they will be a myriad of devices (such as display screens, input devices, home appliances) all communicating via whatever medium is best suited to their application. It is this change in focus that requires better and more complete models if a full understanding of the area is to be obtained.

Chapter 8 - References

- [1] "The Communications Market 2004 – Telecoms", Ofcom Report, August 2004
- [2] "Consumer Products Interact with HAVi", R. Wendorf, et. al, EETimes, 14 June 2001
- [3] "Home Networks Challenge Vendors", B Cole, EETimes, 11 Oct 1999
- [4] "Gateways Bridge Gap Between Home", M. Macaluso, EETime, 11 Oct 1999
- [5] "Home Networking Market Feasibility Study", Jim Mellon, Tality Internal Report, 25th September 2001
- [6] Home Phoneline Networking Alliance, <http://www.hpna.org>
- [7] HomePlug Consortium, <http://www.homeplug.org>
- [8] "Overview of Other PLC Technologies", Duncan McLaren, Tality Internal Report, 5th September 2001
- [9] Universal Plug-n-Play Forum, <http://www.upnp.org>
- [10] HAVi, <http://www.havi.org>
- [11] "A Dynamic Network Scenario Emulation Tool", D. Herrcher, K. Rothermel, Computer Communications and Networks, Oct 2002, p 262-267
- [12] "Physical- and Link-Layer Modelling of Packet-Radio Network Performance", P. McKenney, P. Bausbacher, Military Communications Conference, Oct 1990, p596-602
- [13] "Physical- and Link-Layer Modelling of Packet-Radio Network Performance", P. McKenney, P. Bausbacher, IEEE Journal on Selected Areas in Communications, Jan 1991, p59-64
- [14] "Performance of Reliable Transport Protocol over IEEE 802.11 Wireless LAN: Analysis and Enhancement", H. Wu, Y. Peng, K. Long, S. Cheng, J. Ma, 21st IEEE Computer and Communications Societies Conference, June 2002, p599-607
- [15] "Performance Analysis of the Data Link Layer in the IEC/ISA Fieldbus by Simulation Model", S.H. Hong, S.G. Lee, Proceedings of the 1996 Conference on Emerging Technologies and Factory Automation, Nov 1996, p593-601
- [16] "Comparing Multicast Protocols in Mobile Ad hoc Networks", R. Durst, K. Scott, M. Zukoski, C. Raghavedra, IEEE Proceedings of 2001 Areospace Conference, Mar 2001, p3/051-3/1063

- [17] "Modeling and Evaluation of Bluetooth MAC Protocol", C. Corderio, D. Sadok, D. Agrawal, 10th Intl Conf on Computer Communications and Networks, Oct 2001, p518-522
- [18] "Modelling and Simulation of ATM/BISDN Enterprise Networks", H. Akhtar, MILCOM International Conference, Oct 1994, p87-91
- [19] "An Accurate and Effective Physical Layer Simulator Micro- and Pico-Cellular Radio Systems and Networks", C.A. Santillan, S. Safavi-Naeini, IEEE 2002 Symposium on Antennas and Propagation, June 2002, p 660-663
- [20] "The Design and Analysis of the AFATDS Communication Networks using Simulation", D. Thuente, C. Brown, T. Borchelt, E. Hill, Proc. of the 1996 Tactical Communications Conference, Apr 1996, p267-279
- [21] "Towards a Hybrid Network Model for Wireless Packet Data Networks", H-Y. Hsieh, R. Sivakumar, Proc 7th Intl Symposium on Computers and Communications, July 2002, p264-271
- [22] "Simulation of Multipath Arrival Times for Wireless Indoor Networks", X. Li, P. Flikkema, Proceedings of the IEEE Southeaston '96, Apr 1996, p492-495
- [23] "Indoor Channel Modeling at 60GHz for Wireless LAN Applications", N. Moraitis, P. Constantinou, 13th Intl Symposium on Personal, Indoor and Mobile Radio Communications, Sep 2002, p1203-1207
- [24] "Simulated Performance of the HiperLAN/2 Physical Layer with Real and Statistical Channels", Doufexi, A.; Butler, M.; Armour, S.; Karlsson, P.; Nix, A.; Ball, D.; 2nd Intl Conference on 3G Mobile Communication Technologies, Mar 2001, p407-411
- [25] "Modeling Heterogeneous Sources on Multiple Time Scales", E. Saulnier, K. Vastola, Proc IEEE 15th Conference of the IEEE Computer Societies, Mar 1996, p505-512
- [26] "Simulation Software for Communications Networks: The State of the Art", A. Law, M McComas, IEEE Communications Magazine, Vol 32, Issue 3, Mar 1994, p44-50
- [27] "Advances in Network Simulation", L. Breslau, D. Estrin, K. Fall, S Floyd, J. Heidemann, et. al, IEEE Computer, Vol 33, Issue 5, May 2000, p 59-67
- [28] OpNET Home page: <http://www.opnet.com/>
- [29] "Generic Approach to LAN Modelling", P. Kavi, V. Frost, K. Shanmugan, Proc 1991 Winter Simulation Conference, p716-724

-
- [30] "MAC throughput analysis of HomePlug 1.0", M.H Jung, et. Al.
IEEE Communications Letters, Volume 9, Issue 2, Feb. 2005, p184 - 186
- [31] "Improving HomePlug Powerline Communications with LDPC Coded OFDM", C. Hsu, et al., 28th Annual Intl. Telecommunications Energy Conference, Sept 2006, p1-7
- [32] "Power lines as high speed data transmission channels: Modelling the physical limits", K. Dostert et al. IEEE 5th International Symposium on Spread Spectrum Techniques and Applications, vol. 2, sep 1998, pp. 585–589
- [33] "A Powerline Communication Network Infrastructure for the Smart Home", Yu-Ju Lin et al., IEEE Wireless Communications, Vol 9, Issue 6, Dec 2002, p104-111
- [34] "Performance Analysis of HomePlug 1.0 MAC with CSMA/CA", Min Young Chung et al., IEEE Journal on Selected Areas in Communications, Vol 24, Issue 7, July 2006, p1411-1420
- [35] "Contention Window based Parameter Selection to Improve Powerline MAC Efficiency for Large Number of Users", K. Tripathi et al., 2006 Intl. Symp. On Power Line Communications, March 2006, p189-193
- [36] "Improving the Data Transmission Throughput over the Home Electrical Wiring", M.E.M. Campista et al., IEEE Conference on Local Computer Networks, Nov 2005, p318-327
- [37] "Efficient Framing and ARQ for High-Speed PLC systems", S. Katar et al., 2005 Intl Symp on Power Line Communications, April 2005, p27-31
- [38] "Periodic Contention-Free Multiple Access for Broadband Multimedia Powerline Communication Networks", Yu-Ju Lin et al., 2005 Intl Symp on Power Line Communications, April 2005, p121-125
- [39] "HomeMAC: QoS-based MAC Protocol for the Home Network", Woo-Joo Hwang, et al., 2002 Intl. Symp on Computers and Communications, July 2002, p407-414
- [40] "Turbo-coding and Bit-loading Algorithms for a HomePlug-like DMT PLC System", S. Morosi, et al., 2006 Intl. Symp. On Power Line Communications, March 2006, p227-231
- [41] "Rapid Prototyping of Hardware Systems via Model Reuse", L. Chaount, S. Garin, A. Vachoux, D. Mlynek, 8th Intl Workshop on Rapid System Prototyping, June 1997, p150-156

- [42] "Hardware Simulation with Software Modelling for Enhanced Architecture Performance Analysis", B. Kadrovach et. al., Proc 1998 National Aerospace Conference, July 1998, p454-461
- [43] "A Comparative Study of Modeling at Different Levels of Abstraction in System on Chip Designs: A Case Study", S. Jayadevappa et. al., Proc IEEE Computer Society Annual Symposium on VLSI, Feb 2004, p42-58
- [44] "SystemC and the Future of Design Languages: Opportunities for Users and Research", G Martin, Proc 16th Symposium on Integrated Circuits and System Design, Sept 2003, p61-62
- [45] "The Modelling of Embedded Systems Using HASoC", P. Green, M. Edwards, Proc 2002 Design, Automation and Test in Europe Conference, Mar 2002, p752-759
- [46]
- [47] "Modelling and Simulation with Hardware Description Languages", J. Armstrong, 10th Annual IEEE Intl ASIC Conference, Sept 1997, p329-334
- [48] "A Comparative Study of Modeling at Different Levels of Abstraction in System on Chip Designs: A Case Study", S. Jayadevappa et. al., Proc IEEE Computer Society Annual Symposium on VLSI, Feb 2004, p42-58
- [49] "Using the C Language to Reduce the Design Cycle of an MPEG-2 Video IC: A Case Study", C. Shi, H. Shenghua, Y. Lien, 2nd Intl Conference on ASIC, Oct 1996, p364-367
- [50] "Synthesis of Hardware Models in C with Pointers and Complex Data Structures", L. Semeria, K. Sato, G. De Micheli, IEEE Transaction on VLSI Systems, Vol 9, No 6, Dec 2001, p743-756
- [51] "System Level Design Using C++", D. Verkest, J. Kunkel, F. Schirrmeister, Proc Design Automation and Test in Europe Conference, Mar 2000, p75-81
- [52] "C++ Based System Design of a 72 Mb/s OFDM Transceiver for Wireless LAN", D. Verkest, W. Eberle, P. Schaumont, et. al, IEEE Conference on Custom Integrated Circuits, May 2001, p433-439
- [53] "Design of a JBIG Encoder with SpecC Methodology", J. Peng, IEEE Intl Symposium on Circuits and Systems, May 2001, p5.4.1-5.4.6
- [54] "Design of Real-Time Emulators of Electromechanical Systems", S. Saoud, D. Gajski, Intl Conference on Power System Technology, Oct 2002, p828-833

-
- [55] "The Standard SpecC Language", M. Fujita, H. Nakamura, 14th Intl Symposium on System Synthesis, 2001, p81-86
- [56] "Image Processing Algorithms on Reconfigurable Architecture using HandelC", V. Muthukumar, D. V. Rao, Euromicro Symposium on Digital System Design, Sept 2004, p218-226
- [57] "Modelling Digital Systems Using VHDL", P. Ashenden, IEEE Potentials, Vol 17, Issue 2, May 1998, p27-30
- [58] "C-Based SoC Design Flow and EDA Tools: An ASIC and System Vendor Perspective", K. Wakabayashi, IEEE Transactions on Computer-Aided Design of ICs and Systems, Vol 19, No 12, Dec 2000, p1507-1522
- [59] SystemVerilog Homepage, <http://www.systemverilog.org>
- [60] "Hardware Architecture Modelling using an Object-Oriented Method", F. Mallet, et. al., Proc 24th Euromicro Conference, Aug 1998, p147-153
- [61] "C++ Based System Design of a 72 Mb/s OFDM Transceiver for Wireless LAN", D. Verkest, et al., 2001 IEE Conference on Custom Integrated Circuits, May 2001, p433-439
- [62] "System Level Design Using C++", D. Verkest, et al., Proceedings 2000 Design Automation and Test in Europe Conference, March 2000, p74-81
- [63] SystemC Homepage, <http://www.systemc.org>
- [64] "Object-Oriented High Level Synthesis Based on SystemC", E. Grimpe, F. Oppenheimer, 8th Intl Conference on Electronics, Circuits and Systems, Sept 2001, p529-534
- [65] "Formal Verification of a Bus Structure Modelled in SystemC", A. Habibi, et. al, 2nd Annual IEEE Northeast Workshop on Circuits and Systems, June 2004, p61-64
- [66] "High-Level System Modelling and Architecture Exploration with SystemC on a Network SoC: Case Study", H.O. Jang et. al., Proc 2004 Design, Automation and Test in Europe Conference, Feb 2004, p538-543
- [67] "Native ISS-SystemC Integration for the Co-Simulation of Multi-Processor SoC", F. Fummi, et. al, Proc. 2004 Design, Automation and Test in Europe, Feb 2004, P564-569
- [68] SpecC Homepage, <http://www.specc.org>

-
- [69] “Teaching System-Level Design using SpecC and SystemC”, R. Walstrom, et. al., IEEE Intl Conference on Microelectronic Systems Education, June 2005, p95-96
- [70] “The Standard SpecC Language”, M. Fujita, H. Nakamura, 14th Intl Symposium on System Synthesis, 2001, p81-86
- [71] “High Level Synthesis of Timed Asynchronous Circuits”, T. Yondea et. al., Proc 11th IEEE Intl Symposium on Asynchronous Circuits and Systems, Mar 2005, p178-189
- [72] “System-Level HW/SW Co-Simulation Framework for Multiprocessor and Multithread SoC”, M. Chung, et. al., 2005 Intl Symposium on VLSI Design, Apr 2005, p177-180
- [73] “Design of Real-Time Emulators of Electromechanical Systems”, S. Saoud, D. Gajski,
- [74] “Formal Models for Embedded System Design”, M. Sgroi, L. Lavagno, A. Sangiovanni-Vincentelli, IEEE Design and Test of Computers, Vol 17, Issue 2, June 2000, p14-27
- [75] “Towards a New Standard for System-Level Design”, S. Liao, Proc 8th Intl Workshop on Hardware/Software Codesign, 2000, p2-6
- [76] “Hierarchical Modelling in the Simulation of Electronic Circuits”, R. Zlatanovici, et. al., Proc 1998 Intl Semiconductor Conference, Oct 1998, p497-500
- [77] “Software Architecture of Universal Hardware Modeller”, N. Kelly, H. Stump, Proc of the Design Automation Conference, Europe, Mar 1990, p573-577
- [78] HomePlug Powerline Alliance, HomePlug 1.0.1 Specification, 2001
- [79] “Powerline Telecommunications (PLT); Coexistence of Access and In-House Powerline Systems”, ETSI Standard TS 101 867, 2000
- [80] “Table of Frequency Allocation”, FCC Standard, 2001
- [81] “Power Line Communications: state of the art and future trends”, N. Pavidou et. al., IEEE Communications Magazine, Apr 2003
- [82] “Narrowband, Low Data Rate Communications on the Low-Voltage Mains in the CENELEC Frequencies – Part I: Noise and Attenuation”, D. Cooper et. al., 2002
- [83] “A Multi-path Model for the Powerline Channel”, M. Zimmerman et. al., IEEE Trans on Communications, Apr 2002

- [84] “A Multi-Path Signal Propagation Model for the Power Line Channel in the High Frequency Range”, M. Zimmerman et. al., Proc 3rd Intl. Symp. on Power-Line Communications and its Applications, Mar 1999
- [85] “A Transmission Line Model for High-Frequency Power Line Communication Channel”, H. Meng et. al., 2002
- [86] “An Analysis of the Broadband Noise Scenarios in Powerline Networks”, M. Zimmerman et. al., Proc 4th Intl. Symp. on Power-Line Communications and its Applications, Apr 2000
- [87] “Modelling and Evaluation of the Indoor Power Line Transmission Medium”, F.J. Canete et. al., IEEE Communications Magazine, Apr 2003
- [88] “Characterisation and Modelling of In-Building Power Lines for High-Speed Data Transmission”, L.T. Tang et. al., IEEE Transactions on Power Delivery, Jan 2003
- [89] “A Universal High Speed Poweline Channel Estimation System”, M. Gotz et. al., Proc. Intl. Zurich Seminar on Broadband Communications, 2002
- [90] “Pthreads Programming”, Bradford Nichols, Dick Buttlar & Jacqueline Proulx Farrell, O’Reilly Publishing, p 38-40
- [91] “Improved Mitchell-based Logarithmic Multiplier for Low-Power DSP Applications”, Duncan McLaren, Proc. IEEE Intl. System-on-Chip Conference, Sept. 2003, p53-56
- [92] “How to draw Nassi-Shneiderman Diagrams”,
<http://www.smartdraw.com/tutorials/software-nassi/nassi.htm>
- [93] “Error Control Coding – From Theory to Practice”, Peter Sweeney, John Wiley & Sons Ltd. Publishing