FPGA DESIGN AND DEVELOPMENT OF A DIGITAL VIDEO BROADCASTING (DVB) BASED CHANNEL ENCODER USING VHDL

CHUAH, Kao Hsiung

A thesis submitted in partial fulfilment of the requirements of Nottingham Trent University for the degree of Master of Philosophy

School of Computing & Informatics

JANUARY 2005

Abstract

This research thesis presents a cost effective implementation of Digital Video Broadcasting (DVB) based channel encoder, using Field Programmable Gate Array (FPGA), for an experimental 42 GHz Multimedia Wireless System (MWS): The Nottingham Trent University Campus Network Trial System.

This thesis details investigations and the subsequent design and testing of a channel encoder for the 42 GHz MWS network trial. This includes identifying an FPGA as the development platform; examining, verifying and implementing off-the-shelf Intellectual Property (IP) cores as part of the encoder design. Control algorithms were designed to ensure reliability of data-flow processes. The channel encoder also reconditions the transport packets, for compatibility between system modules and the IP core. Functional modules were coded separately using hardware description language and finally integrated as a system aided by Electronics Design Automation.

As this channel encoder is part of the DVB-Satellite (DVB-S) physical layer that can be evaluated on the 42 GHz campus network experimented test-bed, standard interfaces between systems were used and the encoder specifications were in compliance with the DVB-S standard, to work with off-the-shelf DVB-S set-top-boxes (STBs). Device input/output electrical characteristics were also investigated and adapted to the system. Taking advantage of the flexibility of FPGAs, a combination of Forward Error Correction (FEC) coding schemes were made available that can be reconfigured to be applied to the radio channel. The final FPGA compilation shows a total of 1,461 logic elements and 15,616 memory bits being used on the Cyclone EP1C6Q240C6 device.

The hardware was tested, operating at 26.666 Mbaud for an FEC code rate of 3/4 and 40.000 Mbaud for an FEC code rate of 1/2. The complete end-to-end system was verified using both emulated and ‘live’ digital television transport multiplex. The status register of a satellite STB was used to confirm its functionality

This research has resulted in an inexpensive implementation of a DVB channel encoder for millimetre-wave broadband fixed wireless access offering television broadcasting and interactive data services. The channel encoder was programmed onto an FPGA and has been effectively tested as part of the campus network trial. Further development anticipates dynamic reconfiguration with adaptive capabilities.

Acknowledgements

This thesis is dedicated to my parents back in Malaysia, for their love, support and understanding throughout the years. No words could describe my appreciation to them for having confidence in me to study and manage life in Nottingham. I would also like to thank my relatives here who have provided me with a sense of security in this foreign place. And all my love goes to Ms. Sharon Cheang, for her patience and motivations that helped me through the final stages of this research.

My most sincere gratitude goes to my supervisors, Dr. Richard Germon and Dr. Steve Clark, for their guidance and encouragement throughout the course of my research, especially during the writing up phase. Many thanks goes to Professor Brian O'Neill for his expert advice and my heartfelt appreciation goes to all technicians and member of staff in the university for their valuable support. Also to my colleagues, Mr. W.Y. Mun, Dr. E.B. Lam, Mr. K.H. Lee and Mr. P.C. Nyam, who were more than just friends, as they have always been there to lend a helping hand.

Last but not least, a big thank you to all my dearest friends, Mr. E.H. Yap, Mr. D. Lonjon, Ms. J.Y. Lee, Mr. K.W. Choy, Ms. J. Wong, and Ms. L.L. Chang, who have been an important part of my life and brightened it with wonderful ideas and crazy antics.

Table of Content

List of Acronyms

AMPP	Altera Megafunction Partners Program
ATM	Asynchronous Transfer Mode
BER	Bit Error Rate
BFWA	Broadband Fixed Wireless Access
BSG	Broadband Stakeholder Group
CAT-5	Category-5
CPLD	Complex PLD
DAM	Data Acquisition Module
DQPSK	Differential QPSK
DSL	Digital Subscriber Line
DTH	Direct-to-Home
DVB	Digital Video Broadcasting
DVB-ENC	DVB Channel Encoder
DVB-PHY	DVB Physical Layer
DVB-RC	DVB Return Channel
DVB-S	DVB Satellite
EDA	Electronic-Design Automation
EDIF	Electronic Design Interchange Format
EEPROM	Electrically Erasable Programmable Read-Only Memory
EHF	Extra-High Frequencies
ERC	European Radiocommunications Committee
EU	European Union
FDIL	FEC: Deinterleaver locked
FDRL	FEC: Derandomizer locked
FIFO	First-In-First-Out
FPGA	Field Programmable Gate Array
FPLD	Field PLD
FVL	FEC: Viterbi locked
FWA	Fixed Wireless Access
GUI	Graphical User Interface
HDL	Hardware Description Language
HFC	Hybrid Fibre Coaxial
IC	Integrated Circuit
IEEE	Institute of Electrical and Electronics Engineering

IF	Intermediate Frequency
INA	Interactive Network Adapter
IP	Intellectual Property
ISO	International Organisation for Standardisation
ITU	International Telecommunication Union
LAB	Logic Block Array
LCM	Least Common Multiple
LE	Logic Element
LMDS	Local Multipoint Distribution System
LNB	Low-Noise Block-downconverter
LPM	Library of Parameterised Modules
LSB	Least Significant Bit
LUT	Lookup Table
MAC	Medium Access Control
MII	Media Independent Interface
MPEG	Moving Picture Experts Group
MPEG-2 TS	MPEG-2 Transport Stream
MVDS	Multipoint Video Distribution System
MWS	Multimedia Wireless Systems
NIC	Network Interface Card
NRE	Non-Recurring Engineering
NTU	Nottingham Trent University
OSI	Open System Interconnection
P2S	Parallel to Serial
PCI	Peripheral Component Interconnection
PCI	Peripheral Component Interconnection
PHY	Physical Layer
PLD	Programmable Logic Device
PLL	Phase Locked Loop
PMC	PCI Mezzanine Card
PRBS	Pseudo Random Binary Sequence
QPSK	Quaternary Phase Shift Keying
RAM	Random Access Memory
RF	Radio Frequency
RS	Reed-Solomon
SDD	Satellite Demodulator and Decoder
SMB	Subminiature B
SoC	System on a Chip

SPI	Synchronous Parallel Interface
SRAM	Static RAM
STB	Set-top box
TCP	Transmission Control Protocol
TDM	Time Division Multiplexing
UK	United Kingdom
VCO	Voltage Control Oscillator
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
XOR	Exclusive-OR

List of Figures

List of Tables

1 Introduction

Telecommunications and broadcasting in the new millennium has emphasised efficient transport capacity at acceptable cost for the 'last mile' connections: the access network [1, 2]. As a result of evolution in on-demand interactive multimedia contents and digital broadcasting services, demands for broadband capacity have increased at an incredible rate.

Existing service providers facing the broadband requirement issue are quick to improve existing technologies and develop new systems to meet the challenges of convergence in telecommunication and broadcast services. Broadcast networks are developed to have a return link, to facilitate two-way networking. Digital TV services can then offer access to the Internet. Concurrently, telecommunication networks are capable of offering high-speed Internet services that are capable of streaming 'live' broadband internet television [3].

Broadband strategies initiated by governments, such as eEurope 2005 [4] within the European Union (EU) and UK Online [5] in the United Kingdom (UK), has emphasised the digital switchover of television broadcasts and broadband access for all citizens. These action plans have stimulated research and development as well as the rapid adoption of new technologies. This is especially so in rural areas, where existing technologies fail. Emerging technologies, such as Broadband Fixed Wireless Access (BFWA), have the biggest impact on extending broadband coverage: besides offering flexibility, scalability, fast installation and rollout, BFWA technology can get round the distance limitations of Digital Subscriber Line (DSL) and the high costs of cable [6, 7].

1.1 42 GHz Band

Transmitting at millimetre-wave frequencies, BFWA technology can afford huge amounts of bandwidth; allowing data rates in the megabits-per-second range to every user, which is not currently possible in lower frequency spectrum. With high bandwidth, multiplexes of digital television can be broadcast at the same time as the interactive data services. As signals at these frequencies are directional and have a short propagation range, potential frequency reuse [8] allows even more efficient use of the frequency band. Cell-based deployment of the system makes localisation of services and contents possible, as well as increasing the overall capacity of the bandwidth.

In 1998, the European Radiocommunications Committee (ERC) initiated a review on the band 40.5 to 42.5 GHz in Europe, which was allocated for the introduction of Multipoint Video Distribution Systems (MVDS) back in 1996, to promote the use of digital technologies and to provide a viable means of delivering interactive services. At that time, MVDS was to provide an alternative method for localised broadcast of television programmes, particularly to areas uneconomical to cable [9]. The review followed the development of interactive multimedia services, for instance, interactive television and broadband internet. Such services demand not just broadcasting channels; but a broadband interactive channel. As a result, Multimedia Wireless Systems (MWS) was introduced through ERC Decision (ERC/DEC/(99)15) in 1999, for the frequency band of 40.5 to 43.5 GHz [10]. The allocated 3 GHz of bandwidth is larger than that of radio, television and cellular telephony combined, at lower frequencies, and is envisaged to sustain the convergence of broadcast and telecommunication services [11].

The 42 GHz band is part of Extra-High Frequencies (EHF) that is not yet widely used for communication systems. This is due to the huge fade effects due to precipitation in the atmosphere. Besides, the signal can only propagate within a limited direct line-of-sight range of 1-3 kilometres [12].

1.2 Cost Effective Multimedia Wireless Systems

MWS encompasses all terrestrial multipoint systems, including telecommunications and broadcasting technology. The service may behave as an access network, a broadcasting service or a combination [13]. The access network is part of Fixed Wireless Access (FWA), which maintains wireless transfer of various kinds of information, including graphics, text, sound, image data and video [14], while its broadcasting service provides delivery of digital TV and radio. When adopting DVB standards, MWS allows network data being carried as part of the digital TV and radio transport system over the same channel [15].

The Nottingham Trent University (NTU) Campus Network Trial is one such MWS deployed in the university that is compliant with the DVB-S standard. The existing broadcasting service transmits terrestrial digital TV signal that is re-modulated in DVB-S format to enable the use of inexpensive direct-to-home (DTH) satellite set-top-boxes (STBs) to decode received signals [16] over the 42 GHz channel. An interactive service was also in place for networked data communications between the base-station and its clients. The system adopts a return link standard that is designed for cable networks. Since the existing system uses professional frequency translation equipment that are expensive, a cost effective replacement hardware system could be developed to provide an inexpensive and robust broadcasting and interactive service over the 42 GHz MWS campus network trial.

This research work is concerned with the implementation of hardware as part of the DVB-based physical layer for the campus network trial system. The hardware is developed in compliance with the DVB-S standard, to enable transmission of digital TV and network data, using the same channel coding scheme and modulation technique. This allows a slightly modified satellite STBs to serve as on-site networking equipment, in addition to decoding digital TV signals, hence eliminating the use of expensive cable networking equipment, to realise a cost-effective MWS at 42 GHz.

As the use of dedicated equipment in the existing system only provides standardised configurations, the DVB-based physical layer, developed in this research, could serve as a platform for applying various coding schemes to the NTU campus network trial. A fixed code rate and Forward Error Correction (FEC) scheme applied means that the link is over protected most of the time due to the link margin that has to be reserved for rain loss [17, 18]. Measurements could be made to characterise a variety of coding methods over the 42 GHz link. These results can potentially be used to devise an adaptive system that changes correspondingly to meet the variable channel conditions to achieve maximum efficiency over the campus network trial [19].

The advancement of programmable devices, such as a Field Programmable Gate Array (FPGA) reduces the costs of developing technologies, such as MWS [20]. The device can be programmed to replace multiple-chip configurations with a single-chip processing solution, making it easily deployable for trials. In the context of this research, expensive and bulky equipment are integrated and replaced with minimum hardware both at base-station and client-sites. With its re-programmable capabilities, modifications to an FPGA design can be quickly made in-house without incurring extra cost [21]. This technology is well-suited for this research work as designs can be prototyped and tested on the campus network trial. In addition, various combinations of coding schemes can be programmed into the device for characterisation work. Newer FPGAs also allow dynamic reconfiguration, where a section or the entire design of the FPGA is reconfigured while operating [22]. This feature provides a means of realising an adaptive DVB-based physical layer for 42 GHz MWS.

1.3 Research Aims

The main focus of this research concentrates on investigations of a hardware system that can be used as a generic platform for applying various coding schemes to the NTU Campus Network Trial. The device is to be developed in compliance with the DVB-S standard adopted by the campus network trial. The development would to take advantage of existing in-house development facilities. Working as a channel encoder with reconfigurable features, this device is to be implemented as part of the DVB-based physical layer that can be evaluated within the campus trial. The objectives are identified as follow,

- Investigation on hardware technologies available in-house to identify a suitable development platform for this research.
- Examine off-the-shelf Intellectual Property (IP) functional cores, such as a Reed-Solomon (RS) encoder and a convolutional interleaver, to identify its suitability for this development. Perform functional simulations to recognise the requirements of these functional cores.
- Develop strategies to incorporate IP functional cores as part of the design of the DVB-S compliant channel encoder.
- Design and develop all modules for DVB-S standard channel encoding processes, such as transport multiplex adaptation, randomization, Reed-Solomon encoding, convolutional interleaving and puncturing.
- Devise algorithms to apply conditioning on the data stream to integrate all modules required by the DVB-S standard on the development platform.
- Perform verification processes to ensure that developed modules are functionally corroborated.
- Carry out end-to-end system tests with the designed DVB-S channel encoder as part of the prototyped 42 GHz MWS campus trial to validate the operations of the completed system.

1.4 Structure of the Thesis

The organisation and brief description of the chapters of this thesis are as follow,

Chapter 2 covers the underlying technology review of the 42 GHz MWS campus network trial, and various services and technologies deployed for the campus trial. An overview of the physical layer is presented with descriptions on DVB-S channel encoding and decoding. This chapter also details the investigation into cost-effective technologies that are involved in the development of the channel encoder, focusing on FPGA technologies and EDA design development with IP cores.

Chapter 3 describes the design and implementation stages of the modules for the DVB-based channel encoder. Design requirements and solutions are presented. Several changes are discussed as modifications were made to the design of the system, such as the inclusion of the 'zero-padding' frame conditioning module and the bypassable design for configurability of the coding processes.

Chapter 4 details the verification processes of the DVB-S channel encoder. The completed modules were first verified individually using HDL software simulations. The simulations results were then correlated with another result taken from simulation software, such as Simulink, that is widely used in the industry. The completed channel encoder was also verified on hardware as part of the prototyped campus trial test. Configurations and outcome of these tests are also presented.

Chapter 5 summarises the research work undertaken and defines further possible research directions.

2 Review of Underlying Technology

The use of standard professional equipment as part of a test platform is often faced with lack of flexibility as it is built with limited customisation capabilities to conform to one existing standard, such as [56]. At the same time, professional equipment built for multiple functionalities and standards is expensive. To explore new applications of MWS, such as to enable an MPEG-based interactive data service to be tested over the relatively new 42 GHz frequency band, a certain level of flexibility is required. Therefore, with the aim of developing a cost effective reconfigurable hardware as part of the physical layer for the 42 GHz MWS, a review of the technologies underlying the NTU campus network trial is given. Details on facilitating technologies, such as FPGA and EDA tools, are also presented.

2.1 Electronic Design Automation

Electronic Design Automation (EDA), also known as electronic computer-aided design (ECAD), is a set of computer aided tasks that are used for design and development of electronic devices such as integrated circuits (ICs) [23, 45]. It uses the top down design approach, where the super ordinate systemic specifications are described on top and detailed assembly modules are described further down on the design. EDA can be applied to the design of the entire system as defined by specifications, down to production of the device, including development of printed circuit boards (PCBs) and the embedded system device driver software [24].

Today, the design of an IC can be a complicated task, especially with large scale system integration and competitive time to market. The use of EDA significantly shortens the development cycle as a computer is used to perform time consuming tasks, such as logic synthesis, simulation and timing analysis. By providing comprehensive and accurate system simulations, the use of EDA also promotes zero-errors development of the hardware. This is important as with a shorter time to market, designs are restricted to the number of revisions prior to the manufacturing stages.

EDA not only increases design productivity and precision, it also protects the Intellectual Property (IP) of the design. With the integration of multiple design components in a single device, such as a full custom Application Specific Integrated Circuit (ASIC), designs are better protected compared to separate ICs wired together on a printed circuit board. On the other hand, generic functional designs, such as a Reed-Solomon encoder core or a microprocessor core, can be designed once and re-used as part of another system. These designs can also be fully documented and sold as IP cores.

EDA has come a long way since early 70s. Powered by high-performance modern personal computers (PCs), EDA tools allow designers to develop, simulate and verify ICs and systems reliably. Besides the traditional graphic-based design entry, the use of standardised hardware description languages (HDLs), such as Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL), Verilog HDL and embedded 'C', as one of the design entry methods is one the best features of EDA that has revolutionised design methodology [24]. These description models written in Register Transfer Level (RTL) are then being compiled and synthesised. These processes translate the description of the design into a gate level netlist. To offer the flexibility of transferring design data between different EDA tools, netlists are usually presented in Electronic Design Interchange Format (EDIF) as it is one of the world's most widely used formats [25].

Design simulation is an important stage of the design flow using EDA. It allows designers to simulate and verify systems efficiently and reliably to ensure that the design has zero-error, before moving on to the next stages of development [26]. Tapping on the computing power of modern PCs, stimuli can be virtually inserted to simulate the design. The simulation output obtained can then be analysed to determine if the design has operated the correctly. With the high complexity of larger designs, test benches or logic waveforms can be applied as a set of stimuli to the system during simulation to test for correct responses of the design.

For designs taking the HDL modelling route, such as this research, the synthesis process automates the implementation stages beyond RTL after all design errors are rectified. It automated processes right down to generating gate-level design files for production of target device. During this stage, RTL descriptions are decomposed into Boolean equations. Then, the automated placement and routing process uses the equations to instantiate logic gates and decide on the location of the blocks based attributes of the target device [24]. Interconnection paths are then made between the logic gates. Delays along signal paths are then calculated identifying total delays of the design and critical paths that limit the performance of the system. Using a computer, static timing analysis can be performed quickly and accurately based on target device. This helps determine if the design is compliant to timing constraints as well as maximum performance of the device. Ultimately, one or more gate-level design files are generated for the target device. This design file can be submitted for hardware production. Some proprietary EDA tools for Programmable Logic Device (PLD) and Field Programmable Gate Array (FPGA) development platform also include a device programming tool to download design information into a PLD or FPGA device since programming methods for these devices are exclusively associated with technologies used by vendors.

Design re-use has been the basis of system integration and it is made easier by EDA. As more logic gates can be produced within the same Silicon area, designs can be incorporated into a single ASIC to promote better signal integrity, thus enabling systems to operate at higher frequencies. A single design can be added into libraries of an EDA tool and re-used in other systems to boosts design productivity especially when designing larger systems. Design re-use also allows various commercial IP cores, such as Reed-Solomon encoder and convolutional interleaver cores, to integrate as System-on-a-chip (SoC).

2.1.1 Altera Quartus II Web Edition

The Altera Quartus II is one of many software packages used for EDA. The software is a comprehensive design environment that features everything needed to start designing with Altera devices [27]. This is the latest EDA development software for all of Altera devices replacing the older Max+PLUS II software [28]. It integrates RTL synthesis support for standard Very-High Speed Integrated Circuit Hardware Description Language (VHDL) and Verilog HDL design entry as well as increased design performance for older devices. The software is available as part of the Altera subscription program, but a one-year non-perpetual license is usually included with development kits. A stripped-down version, dubbed Quartus II Web Edition, is available for free with limited support for devices, features and third-party tools.

The web edition of Quartus II software is available for free, with its license being renewable every 150 days. Recent versions of the software can perform automated license renewal over the internet. Although some of the features are disabled, the software provides everything needed to design low-cost Altera devices. A detailed comparison between the full and web edition of Quartus II software can be found in [29].

Quartus II Web Edition software has been targeted for the development of this research. The decision for this software was largely dictated by the targeted hardware, which is the Altera Cyclone family FPGA. As mentioned previously, FPGA architectures are very much tied with specific development software. This is due to the exclusive algorithms and technologies that are used by vendors. With the Altera route, the use of Quartus II software is unavoidable.

As a full-scale programmable device development software, Quartus II provides solutions to all stages of the design. Figure 2-1 shows the complete design flow of Quartus II software as well as its interaction with other third-party EDA tools [30]. The software supports both text and symbol based design entry methods. Designs can be entered using schematic capture or described in Verilog HDL and VHDL. Designs entered using either method can be converted between each other if required. Such flexibility is not available when using third-party HDL tools, such as ModelSim, to perform synthesis and simulations. For the compilation stages of the design flow, Quartus II is required to perform place and route, timing analysis as well as to generate a programming image for the Cyclone family FPGA. By default, these procedures are fully automated to take advantage of its optimisation algorithms. Each logic function will be assigned to the best logic cell location for routing and timing [30].

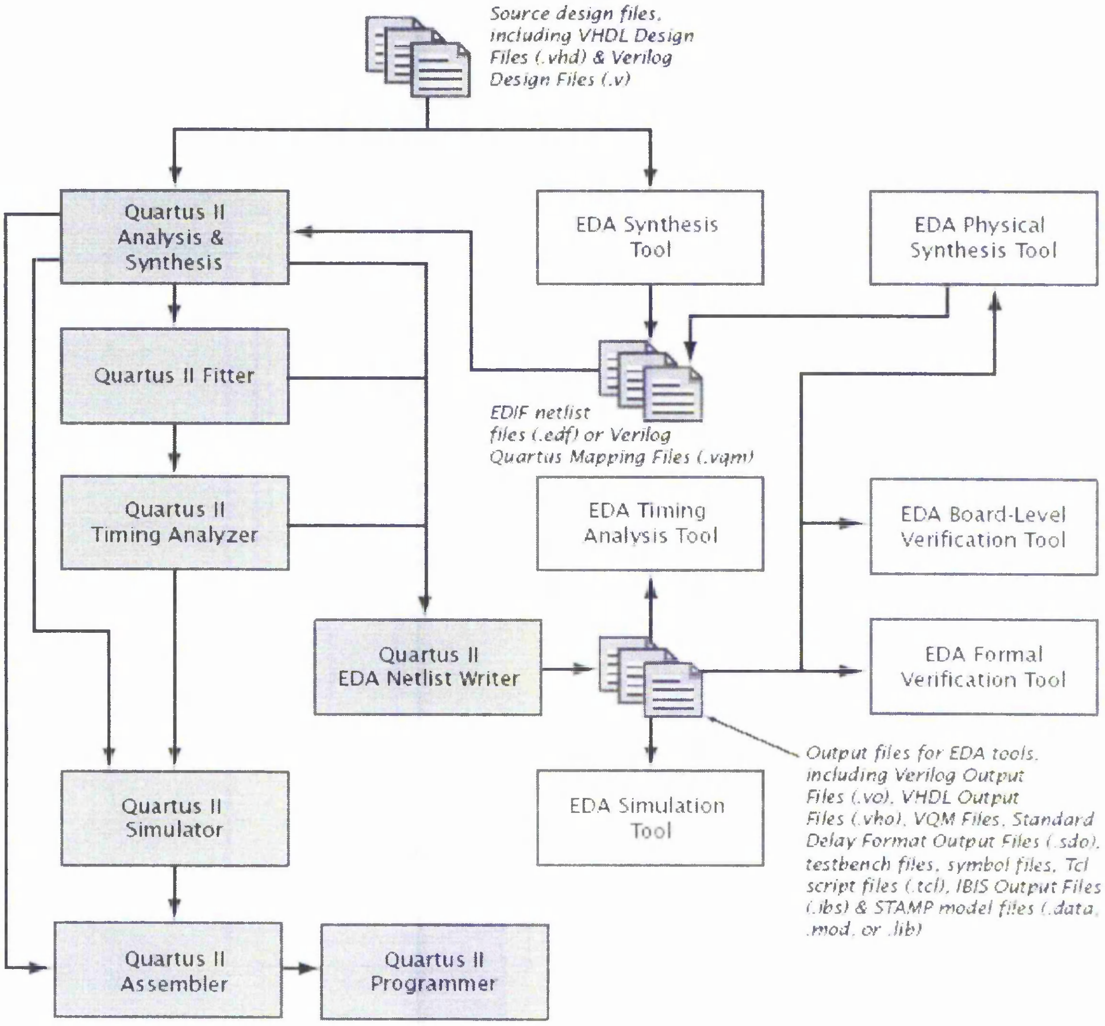

Source design files, including VHDL Design Files (.vhd) & Verilog Design Files (.v)
Quartus II Analysis & Synthesis
EDA Synthesis Tool
EDA Physical Synthesis Tool
Quartus II Fitter
EDIF netlist files (.edf) or Verilog Quartus Mapping Files (.vqm)
Quartus II Timing Analyzer
EDA Timing Analysis Tool
EDA Board-Level Verification Tool
Quartus II EDA Netlist Writer
EDA Formal Verification Tool
Quartus II Simulator
EDA Simulation Tool
Output files for EDA tools, including Verilog Output Files (.vo), VHDL Output Files (.vho), VQM Files, Standard Delay Format Output Files (.sdo), testbench files, symbol files, Tcl script files (.tcl), IBIS Output Files (.ibs) & STAMP model files (.data, .mod, or .lib)
Quartus II Assembler
Quartus II Programmer

Figure 2-1: Quartus II design flow and EDA tool support

Besides offering a full development system and support for the Cyclone family FPGA, the Quartus II Web Edition software also offers support for the IP cores provided by Altera and its Altera Megafunction Partners Program (AMPP) partners. Its support of the OpenCore Plus feature allows time-limited programming files to be generated for hardware evaluation of most cores [31]. IP cores such as the Reed-Solomon Encoder can be used for rapid prototyping.

2.1.2 Third-party Verification

Although Altera Quartus II software is required at later stages of the design flow for hardware designing, third-party verification tools, such as ModelSim [32], can be used at early stages of design using HDLs for functional verification. Such an approach could potentially improve the HDL coding technique and productivity. As verification of design in the early stages is purely functional, issues such as optimisation, and timing delays, can be taken out of consideration. In addition, testbenches can be written and used in functional simulations to automate the verification process. Even so, a verified design is require to be ported into Quartus II software in the later stages for further verification with timing delays and compilation.

However, one disadvantage for using a third-party tool is the additional time required to spend on familiarisation and learning to use the tool efficiently.

2.2 Very High Speed Integrated Circuit HDL (VHDL)

VHDL is adopted as the method of design entry for this research. It has been widely accepted throughout the EDA market as the language to describe and specify a wide variety of electronic designs since initial ratification by the Institute of Electrical and Electronics Engineering (IEEE) in 1987 [33]. This has revolutionised design methodology as traditional schematic-based design used to be part of the design process. Similar to most modern programming languages, VHDL can be easily structured into smaller modules individually based on top-down system design methodology for better design management.

VHDL has been previously the choice of design entry method for FPGA design based projects at NTU. Past experiences in these projects have contributed to the know-how of this method as well as the supporting tools that are required [34, 35]. As a result, facilities and resources within the university are readily available.

One of the advantages of using VHDL design entry methodology for IC design is significant reduction of design cycle. When designs are modelled in RTL, VHDL can be synthesised to generate a gate level design automatically. This eliminates logic and geometrical design tasks when designing an IC as the coded designs are automatically converted into Boolean equations, placed and routed on a target device [24, 33]. Unlike traditional schematic design entry, previously written generic designs can be re-used and implemented as part of another design efficiently using VHDL.

The ability to easily simulate a design is one of the factors VHDL is contributing to overall improved design quality. The use of simulation tools have allowed designs to be verified before being implemented on target device. Test benches can be written to insert stimuli into the design and monitor outputs to verify its operations. Furthermore, with graphical and logic design tasks automated, the use of synthesis tools provides a powerful capability to optimise a design [33]. Most synthesis tools today are capable of optimising designs for speed or for space efficiently.

Designs coded using VHDL is independent of hardware vendor and technology. This is because different libraries are provided with synthesis tools used for different hardware technology, such as ASIC, PLD and FPGA. Therefore, this makes VHDL design entry highly portable. The same VHDL source code written for a design can be synthesised and optimised to implement on a PLD or FPGA for rapid prototyping before being synthesised and optimised to implement on an ASIC.

As the syntax for VHDL is capable of describing a design at many layers of abstraction, designs can be written at the high-level behavioural level, RTL or gate level structures:

- When written at behavioural level, signals such as clocks, or computational operations such as ADD, can be used without details of implementation. This level of abstraction can be quickly simulated to verify design algorithms prior to detailed development and implementation.

- When written at RTL, descriptions of the design goes one step lower from behavioural level. Operations are schedule to correspond to certain clock edges. In addition, operations are described to specific details such as number of bits assigned and Boolean equations. This level of abstraction can provide the greatest gain in design productivity. The design file written at RTL can be synthesised by EDA tools as it abstracts to the specification level, which is describing the functions of the IC.

- When written at gate level, descriptions of the design consist of components and interconnecting signals that are mapped between them. These components can be made up of basic gates, flip-flops or a combination of both. This level of abstraction is analogous to schematic representation of a design, which describes the functions of the design and how it does it.

2.3 Intellectual Property (IP) Core

One of the advantages of designing with EDA tools is that it is not limited to implementing own designs. Most hardware vendors such as Altera and Xilinx have also developed logic designs with specific functions that are optimised for the target device. In the context of Altera, they are called megafunctions. While some of these basic megafunctions are provided for free as part of the Quartus II megafunction plug-in libraries, other highly specialised IP cores, such as the RS encoder, are sold or licensed with a price tag.

Two main reasons for the rising popularity of using IP cores in designs are reduction in design time and overall cost saving. As chip design is expensive and demands for shorter time-to-market grow, the use of IP cores in designs is essential [36]. Although the use of IP cores involves additional costs for licensing, the resulting reduction in design time can lead to lower overall costs [37]. In addition, purchased IP cores are usually licensed for a long period of time with unlimited use. For example, the Altera's RS encoder megafunction would be licensed for 30 years with unlimited number of times it can be used [42].

As most IP cores in the market are designed to facilitate drop-in instantiation, the design source codes are typically encrypted. Modifications of the design source code are not possible should there be slight changes in design constraints and parameters. For that reason, the IP cores need to be evaluated thoroughly using features such as OpenCore Plus by Altera megafunctions to ensure a careful selection from the start [38].

2.3.1 Altera Intellectual Property (IP) Megafunctions

Altera and AMPP partners offer a wide selection of parameterised megafunctions that are optimised for Altera devices [39]. Working with Quartus II software, the optimisation process usually involves setting compilation and synthesis options to maximise density and performance of the IP core. They are also fine tuned to be as fast and as small as possible [36]. These IP megafunctions are tested to ensure flawless implementation and dependability based on Altera's "AMPP Approved" qualification process [40].

Altera's IP megafunctions were targeted following the Altera approach for design software and FPGA device. Together with Quartus II software, one of the main benefits of using the IP megafunctions is the OpenCore Plus feature. This feature allows the megafunction to be simulated with the rest of the design to verify the overall functionality of the design. In addition, time-limited programming files for the design, including the megafunction, can be generated to be programmed into a device for hardware verification [41]. This feature is available for most megafunctions offered by Altera and AMPP partners to facilitate hardware evaluation to perform board level design verification before deciding to purchase a license [31]. Under the Altera University Program, the RS encoder can be licensed at a special price of US$199.50, which is 90% cheaper than the regular price [42]. When purchased, the megafunction will be licensed for 30 years and no limit to the number of times it is used.

In the context of this research, the megafunction provides a quick and dependable design solution for standard functional blocks, such as the RS encoder. The ability to have a complete design with the megafunction programmed into a prototype device, without having to purchase a license for the megafunctions, has allowed the system to be tested from end to end. With the megafunction capable of running up to 30,000,000,000 clock cycles, enough time is granted to perform several tests on the system. Therefore, purchase of license for the RS encoder megafunction is not required throughout the period of this research.

2.4 Field Programmable Gate Array (FPGA)

FPGAs started off as 'glue-logic', a simple device that is used to connect other complex logic circuits together. The programmable device integrates several logic devices to reduce chip count and therefore simplifying board design. Today, the role of FPGAs has evolved to implementing complex functional systems such as embedded microprocessors [43]. Static random-access memory (SRAM) FPGAs and complex programmable logic devices (CPLDs) are two of the most common programmable application-specific integrated circuits (ASICs) available on the market, but other FPGA technology such as non-volatile FPGAs and antifuse FPGAs are increasingly popular and contributing to the diversity of programmable ASICs [44].

The smallest unit of logic in an SRAM FPGA is the logic element (LE). It consists of a lookup table (LUT) and a programmable register in the SRAM cells. Usually, multiple LEs are combined to form a configurable logic block, also known as logic block array (LAB). Unlike a CPLD that programs on electrically erasable programmable read-only memory (EEPROM) transistors, the FPGA uses the SRAM that is volatile, thus making it easily reconfigurable. However, with the line between CPLDs and FPGAs blurring in recent years, both types of programmable ASICs are grouped together as FPGAs [45]. Latest CPLDs have dropped the traditional macrocell architecture and opted for LUT configuration, whilst new generation FPGAs now offer non-volatile, instant-on reconfigurable logic [46].These devices keep on increasing in speed and density, while decreasing in manufacturing cost.

The advancement of today's FPGAs can provide high performance data processing capability required in digital signal processing (DSP) applications and high-speed data transfers [47]. With the flexibility to reconfigure, FPGAs are the best target device for development of next generation applications [48]. The advantages of system development on FPGAs include:

Low-cost by design

No Non-Recurring Engineering (NRE) charges are required for development of systems targeting FPGAs. NRE charges are the cost of work done by vendors, including logic design support, chip layout, mask generation, test generation and other services, that a full custom ASIC development requires. On the contrary, a design is programmed into an FPGA via a low cost programming device, which can be performed in-house. Eliminating such costs with FPGA development therefore eliminates the risks of a custom IC development.

Flexibility

Design modifications on FPGAs can be made quickly and without incurring penalty charges. Instead of being custom manufactured as a full custom ASICs, FPGAs are programmed electrically. Modifications on design can be performed in-house, within seconds, using a programming device. In contrast, every design change on a full custom ASIC would incur hefty charges and delays as the process involves manufacturing of a new custom mask. Avoiding such development spending therefore reduces the expenses of developing a custom IC.

Reconfigurability

The FPGA development platform has low inventory risk as the device is only considered to be manufactured when it is programmed with the design. Further to that, the device can be reconfigured for different functionalities and designs [43] whereas the functionality and design for a full custom ASIC is permanent once manufactured at the vendor. In terms of design, generic designs and algorithms can be reused and re-implemented to a new application. With such low inventory risk and reusability, the FPGA platform makes a suitable prototyping tool.

Fast design verification

Designs on FPGAs can quickly have their functionality verified and timing characteristics known, as timing models for a specific FPGA are usually known in advance as timing analysers are included modern synthesis tools. FPGAs are manufactured as soon as they are programmed in-house, while full custom ASICs require manufacturing processes that take weeks or months. With the capability of instant hardware verification, modifications to correct a design flaw can be quickly and easily done. The short turn around time for design modification processes makes the FPGA platform most suitable for rapid prototyping that leads to a shorter time-to-market.

With respect to this research, the aim of developing a generic DVB-S channel encoder can be achieved by using an FPGA. A small quantity of the channel encoder could also be easily reproduced for deployment to client-sites without incurring additional costs except to purchase the FPGAs. With the recent launch of inexpensive FPGAs, such as the Altera Cyclone and Cyclone II family, better performance and higher density FPGAs are offered at lower cost.

From the flexibility attained by targeting the development on an FPGA, another aim of the research can be achieved. Besides being programmed as a completed product, the same FPGA device is a prototyping tool in its own rights. Designs of the system can be altered and programmed onto FPGAs for experimentation. Such flexibility allows standardised configurations to be manipulated and tested on the system to explore new schemes and functions, something not possible on a standard device, such as the Newtec NTC/2080 DVB Modulator [49].

Reuse of modules and systems designed using EDA tools can offer high level of system integration. For example, integration of an Ethernet data adapter module with a DVB-S channel encoder in a single device can offer encapsulation of Ethernet data stream for transmission over a DVB-S channel. Such integration not only provides better overall system performance but also reduces its cost.

The use of EDA tools for FPGA development not only provides synthesis, compilation and simulation with accurate timing characteristics, it also supports the use of IP cores as part of the encoder system design. Highly complicated functional modules, such as the Reed-Solomon encoder can be implemented for rapid prototyping. Although these IP cores can be expensive to use, the development cycle of the system is effectively shortens.

In today's market, Altera and Xilinx are two leading competitors for SRAM FPGA devices, each offering different architectures and advantages. Other vendors such as Actel and Lattice are more popular with non-volatile and antifuse FPGAs [44]. However, the choice of solution is determined by the availability of in-house hardware and software. The Altera route was the most feasible approach, as the university is part of the Altera University Program [50]. In-house availability of Altera's Quartus II EDA software and Altera's FLEX10KE PCI Development Board was already known at the start of this research and further acquisition of Altera Cyclone family FPGAs was made during the course of this research. One of the advantages being part of Altera's university program is the special purchase of Altera devices, design software and IP cores at very low prices. For example, licensing prices for several IP cores were quoted at 90% discount off published price when enquiries were made under the university program.

2.5 FLEX10KE PCI Development Board

The FLEX10KE peripheral component interconnection (PCI) development board was used at the start of this research for prototyping purposes as it was available in-house. The board uses the Altera FLEX10KE family CPLD device. Although this device is referred as a CPLD, it is a volatile SRAM-based device and must be re-programmed at power-up, as the configured data is lost when power is switched off. The board, as featured in Figure 2-2, is well equipped for PCI-based development, especially with a computer. It provides PCI front-end reference design in the form of Altera's megafunction. Besides, the board also comes with additional memory modules [51]. Although the FLEX10KE development board presents all the advantages for PCI prototyping, several shortcomings prevented further development of this research on this platform.

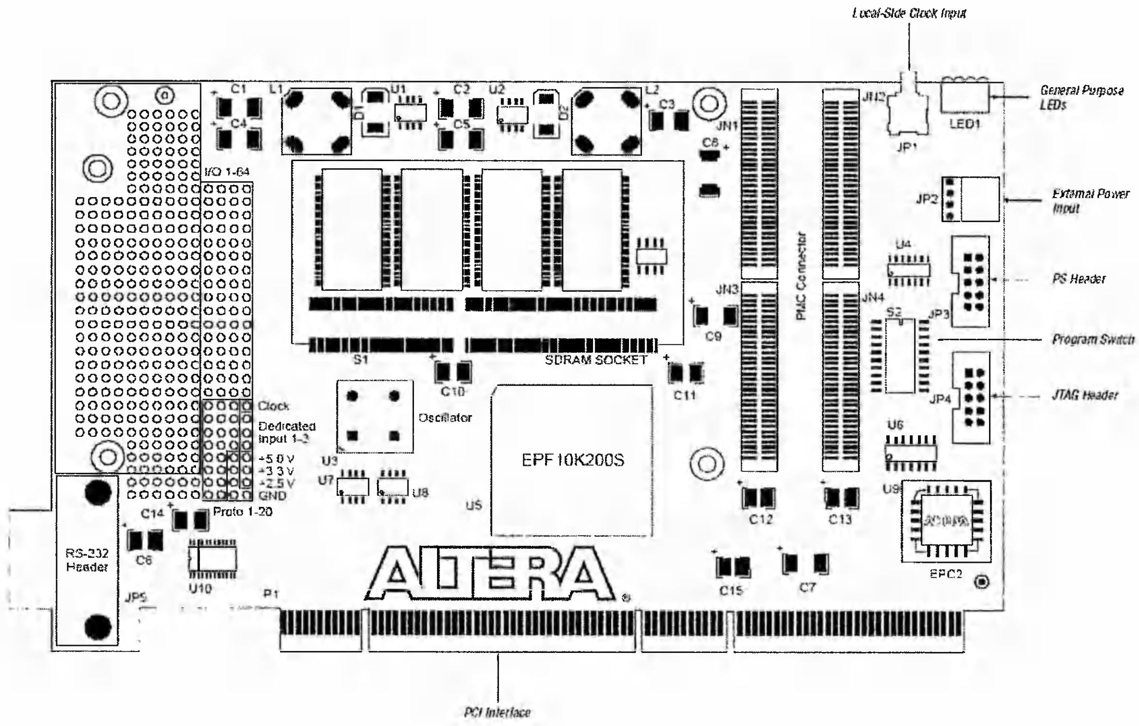


Figure 2-2: FLEX 10KE PCI Development Board

The development board was not supplied with an on-board oscillator, although the solder pads are available. Further investigation revealed that the clocking option for the oscillator was disabled and clocking sources were limited to a system clock via PCI port and external clock via snap-mount Subminiature B connector (SMB). Both sources were found to have limited feasiblity as the design is not PCI-based, and SMB connectors were rare, expensive and not available in-house.

The development board also provided limited on-board input/output (I/O) pins for prototyping. An expansion of its prototyping capabilities can be done by using an additional board using PMC connectors. However, it was found to be not feasible as PMC connectors were rare, expensive and would not be obtained at unit quantity.

2.6 Altera Cyclone family

Introduced in December 2002, the Altera Cyclone family has since been the lowest-priced FPGA family in its class. Targeted for cost-conscious hardware development projects such as this research work, the FPGA family offers almost half the cost of competing low-cost FPGAs per 1000 LEs [52]. Benchmark results also showed that the Cyclone family FPGAs outperform Xilinx Spartan-3 family by an average of 70.2%, when comparing the fastest speed grade devices [53]. Moreover, the web edition of Quartus II software supports the FPGA family at no cost.

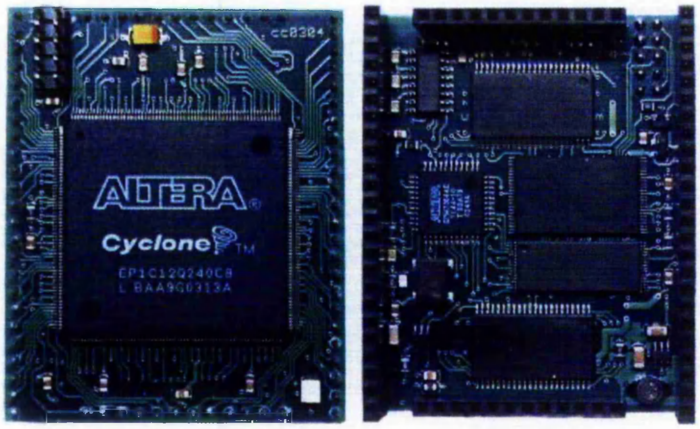
One of the strongest features of Altera's Cyclone family is the facility to implement up to two phase-locked loops (PLLs) per device. The full-featured PLLs can be used for on and off-chip timing management. By using the Altera Quartus II software, the PLLs and their features can be enabled on Cyclone family FPGAs without using any external devices [54].

2.6.1 Cyclone Development Board

The Cyclone development board was acquired by the research group to upgrade its in-house prototyping facilities. Produced by JOP Design, the development board is built for system-on-a-programmable-chip (SOPC) solutions using a Cyclone family FPGA. This mean that the development board is capable of implementing processor core, logic and memory in one device. Extending the memory capabilities of the Cyclone device, this board is built with 512 kilo-byte Flash memory for FPGA configuration and application code, 1 mega-byte (MB) of fast SRAM memory as main memory and up to 128 MB of NAND Flash memory for solid state storage.

Amongst the list of best features, the on-board 20 MHz crystal clock makes this development board an important upgrade, eliminating the use of an external oscillator for improved clock management. The clock signal is connected directly to the clock input of the Cyclone PLL as well as the dedicated clock input of the FPGA. The development board is also built with an Altera MAX7000 device to code the FPGA with programming information from a Flash memory moments after power-up. Alternatively, the FPGA can be programmed manually from a PC via the parallel-port using a ByteBlaster communications cable. As shown in Picture 2-1, the 56 general purpose constellation I/O pins around the board can also be used as probing points for prototyping purposes. Equipped with other supporting components such as watchdog and serial interface driver, the board can therefore be used as an FPGA prototyping platform as well as a ready-to-use module for various applications, such as the DVB-based channel encoder [55].

Reuse and migration of designed modules to target the Cyclone FPGA requires minimum effort when using electronic-design automation (EDA) tools, such as the Altera Quartus II software. The design files can be re-synthesised and verified by simulations with Cyclone device settings. Full compilation can then be performed after the I/O pins are re-assigned.



Picture 2-1: Cyclone Development Kit

2.7 NTU Campus Network Trial

The campus network trial has been implemented at Nottingham Trent University (NTU) in collaboration with MMRadioLink Limited (Phillips UK) to provide a platform for network performance evaluation and exploration of new applications on 42 GHz MWS [16]. The trial was set up to serve as an experimental test-bed towards providing a robust channel to deliver broadcast and interactive services as provisioned in the ERC Decision [10].

2.7.1 Broadcast Service

The broadcast service of the NTU campus network trial adopted the DVB Satellite (DVB-S) standard that is used to provide DTH digital TV [56]. Historically, the 42 GHz band was designated for MVDS, for both analogue and digital TV [9]. Proven in previous trials [57], the adoption of DVB-S compliant configuration not only provides a robust channel, it also enables the use of off-the-shelf satellite STBs at receiver sites for reception of the broadcast services.

Two digital TV multiplexes were re-transmitted over the campus trial. As depicted in Figure 2-3, off-air terrestrial digital TV channels were demodulated into an MPEG-2 TS [58], which is then re-modulated in DVB-S. Each of the digital TV multiplex are transmitted over the campus trial using QPSK modulation, a symbol rate of 17.5 MBaud, and a 3/4 FEC code rate [59]. As both systems are DVB compliant and share the common MPEG-2 TS format, interoperability between the terrestrial and satellite TV signals are conveniently seamless.

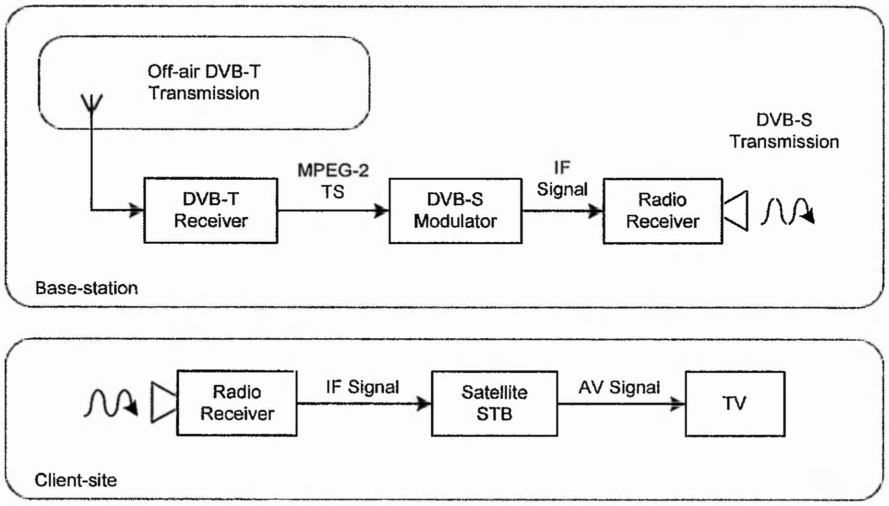


Figure 2-3: Broadcast service

At the client site, the configuration for receive-only broadcast service system is simple. From the millimetre-wave radio receiver, the intermediate frequency (IF) signals are fed directly into a satellite STB. The digital TV multiplex are then demodulated and viewed on a TV set, as depicted in Figure 2-3.

2.7.2 Interactive Service

The interactive service of the NTU campus network trial adopts DVB Return Channel (DVB-RC) [60]. This standard is designed for Hybrid Fibre Coaxial (HFC) networks to provide a return channel for interactivity [59, 61]. At the base-station of the system, an Interactive Network Adapter (INA) is used to control data communications between all connected cable modems in the access network. Network data packets are transported in Asynchronous Transfer Mode (ATM) cells that are encapsulated in MPEG-2 TS frames [62]. Downlink capacity at 13.8 Mbps is shared between all users using time division multiplexing (TDM). The operating frequency of the INA is translated to match the higher IF of the radio transceiver using an interface unit.

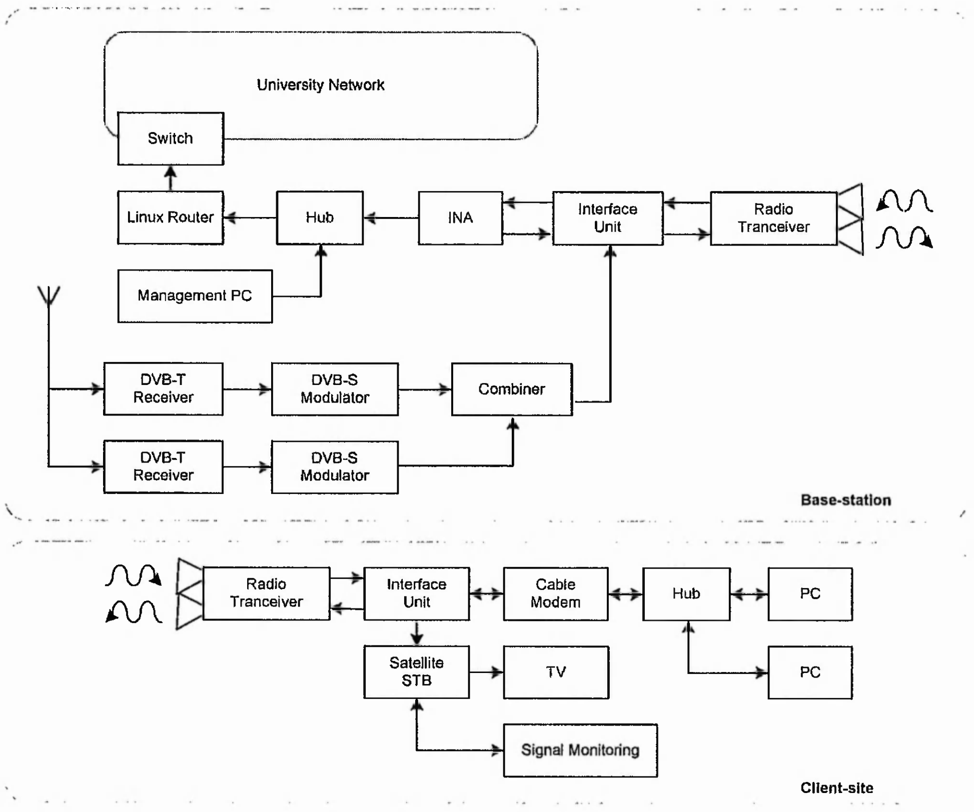


Figure 2-4: Interactive service

At the client location, the uplink data is carried over a 2 MHz channel using differential QPSK (DQPSK). As depicted in Figure 2-4, a cable modem is used to realise the bidirectional broadband data communication. The resulting return capacity of 3 Mbps is shared between users on a time division multiple access (TDMA) basis [59]. Once again, frequency translation is performed between the IF and the operating band of the cable modem.

2.7.3 Prototyped Broadcasting and Interactive Service

The convergence of broadcasting and interactive services is made possible by introduction of a network adapter that encapsulates Ethernet network data transports in

MPEG-2 packets [63, 64]. This is currently being developed at NTU. The combined development of this channel encoder and the network adapter gives rise to changes to the configuration of the interactive service. Such a development would allow the HFC-based interactive link to be replaced by two-way DVB-S links. Implementation of the DVB-S channel coding into re-programmable hardware, such as an FPGA would enable the device to be conveniently deployed at client-sites to provide channel coding of the return link. Frequency Translation between cable equipment and radio system is no longer required.

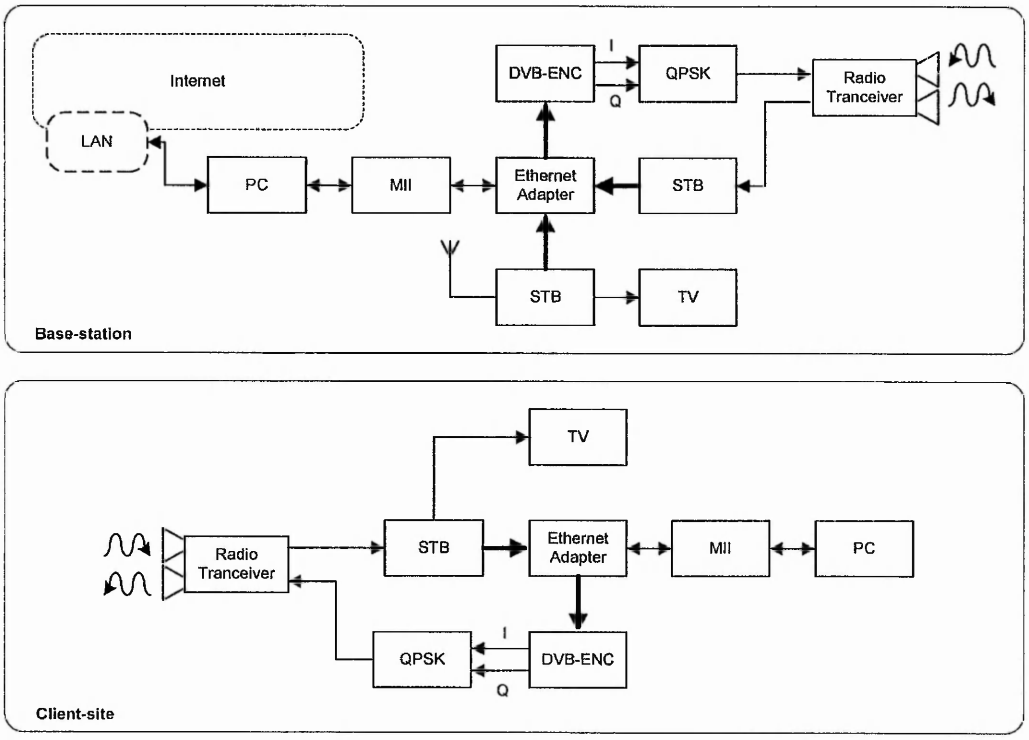


Figure 2-5: Prototyped broadcasting and interactive service

At the base-station, a bi-directional DVB-S link also helps eliminate the use of the INA. The use of a unified standard for both broadcasting and interactive services cuts down instrumentation costs at the base-station as well as at client-sites. The packets are then multiplexed as a part of the MPEG-2 TS and modulated. As depicted in Figure 2-5, the demodulated MPEG-2 TS is connected to a network adapter where the network data is extracted.

2.8 The DVB Physical Layer

The Physical Layer (PHY) provides the physical communication path between two nodes by directly interfacing through the transmission medium [65]. As depicted in Figure 2-6, it is the first layer of the layered protocol model devised by the International Organisation for Standardisation (ISO) for Open System Interconnection (OSI), on which modern network architectures are based. The PHY defines the electrical and mechanical aspects between devices [66]; electrically, they include power characteristics, channel coding and modulation, and mechanically, they include cables, connectors and connector pin assignments.

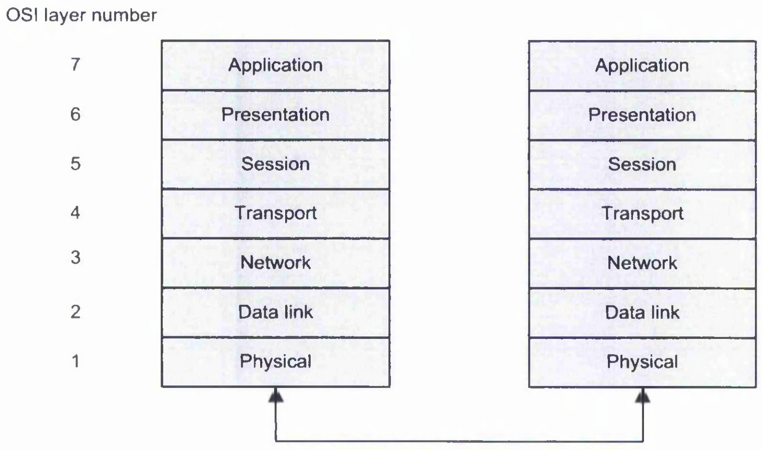


Figure 2-6: The 7 layers of the OSI model

The DVB Project [67] was founded in 1992 to establish the framework for the introduction of MPEG-2 based digital television services. Standards developed by DVB use the MPEG-2 packets as "data containers" to carry various digital contents. These are called packetised elementary streams (PES). As a result of multiplexing, the standard exploits the flexibility of the transport multiplex to offer a variety of TV service configurations, including sound and data services. The transport multiplex, also known as the MPEG-2 transport stream (TS), is then coded and modulated prior to transmission.

The DVB PHY is adopted in this research work on 42 GHz MWS for its robustness and reliability. Such robustness is necessary because Internet data that uses standard Transmission Control Protocol (TCP) is very sensitive to packet loss and non-congestive delays. TCP throughput can drop to between 20%-50% with Internet Protocol packet losses of only 2%, making it almost useless [68, 69]. However, with the large available bandwidths in the 42 GHz frequency band, the system can afford to adopt aggressive channel coding schemes. As with delivering digital broadcasting contents, the DVB PHY employs FEC blocks, such as Reed-Solomon encoder and Convolutional Interleaver, to the network data. These FEC schemes are essentially invisible for TCP.

The DVB-S PHY uses Quaternary Phase Shift Keying (QPSK) modulation. It has the advantage of implementation of existing technologies, particularly chips developed for digital television [12, 56]. With slight modification, it allows the use of commercially available STBs as part of the hardware set-up to get on the network. Such adoption enables services of digital TV as well as broadband network access across 42 GHz MWS using the same set of cost effective hardware.

2.8.1 DVB Synchronous Parallel Interface

Interfaces for devices using MPEG-2 transport packets are defined in [70]. This allows system integration for different applications and interoperability of equipment. An MPEG-2 transport packet usually consists of one MPEG-2 synchronisation (SYNC) byte and 187 data bytes. The MPEG-2 SYNC byte is the first byte of each MPEG-2 transport packet to signal the start of the packet. It carries the value of 47_{HEX}. When the MPEG-2 packet is protected using Reed-Solomon (RS) encoding, the RS protected transport packet would have 16 bytes of additional RS codes concatenated at after the data byte making it a 204-byte packet.

For short distance interfacing, the synchronous parallel interface (SPI) is specified. Two flags were introduced to distinguish 188-byte packets from 204-byte packets, and to signal for valid RS-bytes. With reference to Figure 2-7, data transfer is synchronised to the byte clock. The 8-bit data bus carries the MPEG-2 Transport Steam (TS), while PSYNC signals the beginning of a packet, and DVALID signals valid data bytes.

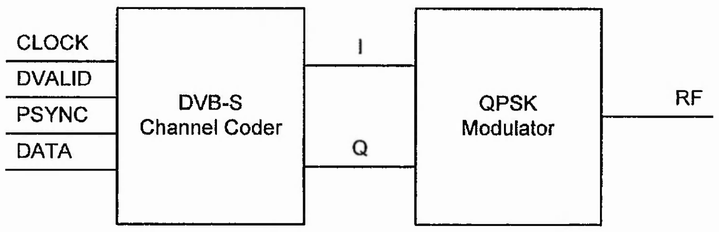


Figure 2-7: DVB Physical Layer

With conventional Gray-coded QPSK modulation employed for DVB-S system, absolute mapping is used [56]. With I and Q, being the in-phase and quadrature channels, the constellation points are simply I, Q, -I, and -Q, corresponding to symbols 00, 01, 11, and 10 [71]. This also corresponds to the output of the puncturing module of the inner coder. Therefore, the I and Q signals can be directly filtered and QPSK modulated.

2.8.2 DVB-S Channel Coding

Channel coding, also referred as error control coding, is used to detect and also correct symbols received in error. Typically, FEC is being coded to incorporate the source signal with additional information to the transmitted data. This information can be used to detect errors and recover the original data when it occurs [65]. A combination of coding algorithms can be applied to minimise the effects of channel noise. For a telecommunication system, these schemes must be standardised to ensure that data can be conveyed over the channel.

EN 300 421 [56] is the standard for satellite services, which employs randomization of the transport stream for energy dispersal, Reed-Solomon encoding for outer coder, convolutional interleaving, and punctured convolutional encoding as inner coder. Figure 2-8 illustrates the channel coding algorithms that are employed by DVB-S standard. The outputs of the channel coder, I and Q, are then QPSK modulated with a carrier signal.

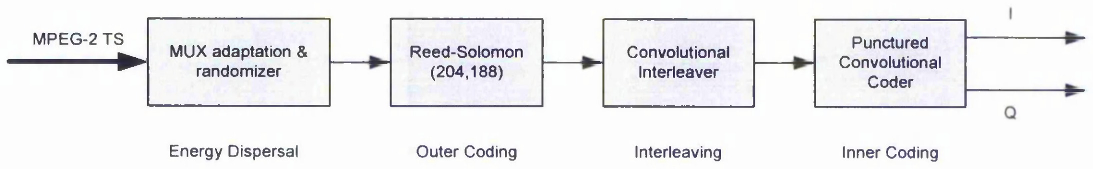


Figure 2-8: DVB-S channel coding functional blocks

2.8.3 Energy Dispersal

In compliance with radio regulations for spectrum occupancy and to ensure adequate binary transitions, the data at the output of MPEG-2 multiplex is randomised. The randomization process, also known as scrambling, spreads any possible concentration of energy at specific frequencies, by applying a Pseudo Random Binary Sequence (PRBS) over the transport packets except at every first byte of the packet, which is the MPEG-2 SYNC bytes [72]. This is to maintain the initiation word to mark the start of the packet. A loading sequence of "100101010000000" is initialised every eighth packet into a PRBS generator that is based on the following polynomial defined in [56]:

$$1 + x^{14} + x^{15} \qquad \textbf{[2-1]}$$

To provide an initialisation signal for the descrambler, the MPEG-2 SYNC byte of the first in every eight transport packet is bit-wise inverted from 47_{HEX} to $B8_{HEX}$. This process is referred to as "Transport Multiplex Adaptation" or SYNC inversion.

2.8.4 Outer Coding

Outer coding is applied at the extreme input and output ends of the transmission chain. The Read-Solomon (RS) code, RS(204,188), which is shortened from the original RS(255,239), is then applied to every randomised transport packet to generate an error protected packet. Each transport packet is fed through the coder byte-wise until the end of the packet is reached. At the same time the values of the RS codes are calculated using coefficients derived from the code generator polynomial, $g(x)$ [56]:

$$g(x) = (x+\lambda^0)(x+\lambda^1)(x+\lambda^2)\ldots(x+\lambda^{15}),\ where\ \lambda=02_{HEX} \qquad \textbf{[2-2]}$$

The RS redundancy bytes concatenated at the end of each transport packet provide protection against burst errors. The code is capable of correcting up to 8 bytes of errors as it adds 16 redundancy bytes at the end of the 188-byte transport packet.

2.8.5 Interleaving

The purpose of interleaving is to increase the efficiency of the RS coding. This is due to the fact that a long burst of errors in transmission can exceed the correction capacity of the RS code. This will cause a failure of RS code to recover the original transport data. The convolutional interleaver is usually an array of buffers where each byte of data is being delayed when read out. As a result of the interleaving process, a long burst of errors during transmission will become scarcely spaced when the stream is de-interleaved at the receiving end [72, 73].

2.8.6 Inner Coding

Inner coding is applied just before the transmission signal is modulated. In the context of DVB-S channel coding, a punctured convolutional code is used. With this coding scheme, signals can be transmitted with at least 5 dB less power than without it, when using QPSK modulation [74]. This reduces transmitter cost and allows increased data rates at the same transmitter power. Based on a rate 1/2 convolutional code, the data stream is fed into a shift register bit-wise. The transmitted signal is obtained by combining various taps at the shift register, based on the constraint length, $K=7$, with generator polynomials coefficients [56, 75]:

$$X = g_1 = 171_{\mathrm{OCT}} = 1111001$$
$$Y = g_2 = 133_{\mathrm{OCT}} = 1011011$$

Puncturing refers to deleting certain bits off the transmission stream in order to raise the overall code rate. This must be done according to the specific pattern standardised in [56]. Several code rates are defined: 1/2, 2/3, 3/4, 5/6 and 7/8, to allow selection of the most suitable level of error correction for a given service.

2.8.7 DVB-S Channel Decoding

One of the main reasons for adopting DVB-PHY for 42 GHz MWS is to enable the use of commercially available STBs in the system. A typical front-end of a STB is made up of a tuner module and the satellite demodulator and decoder (SDD) integrated circuit (IC) [76]. The SDD IC performs demodulation and FEC on the received signal to regain the MPEG-TS data. The FEC algorithms are opposite to that of coding processes and are employed in reverse sequence in relation to the coding scheme. QPSK demodulated I and Q signals provide information to the inner decoder. The signals are de-punctured and decoded by the Viterbi decoder, then de-interleaved, and RS decoded by the outer decoder, as depicted in Figure 2-9. The randomising energy dispersal pattern is removed and the inverted MPEG-2 SYNC bytes are reverted to recover the user data [56].

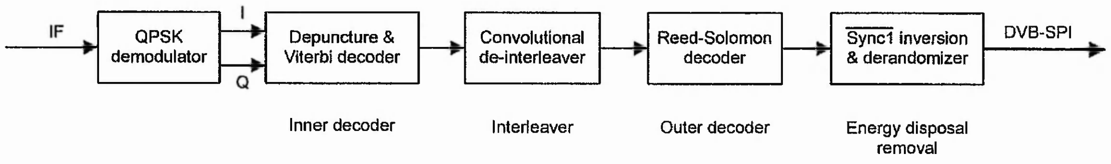


Figure 2-9: Satellite STB front-end block diagram

At the decoding-end, the recovered MPEG-2 TS is then de-multiplexed and de-scrambled by the MPEG-2 source decoder IC into MPEG-2 PESs, as shown in Figure 2-10. A typical digital TV PES is then further decoded into video and audio signals.

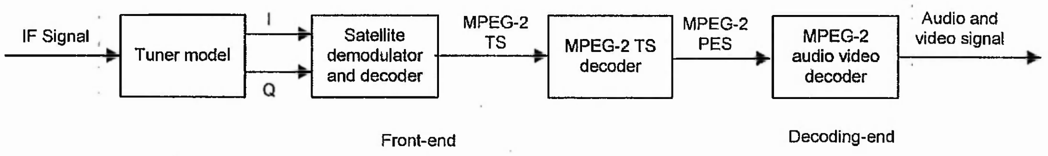


Figure 2-10: Satellite STB front-end and decoding-end

2.9 Humax F1-FOX Set-Top-Box

The satellite STB receiver used in the campus network trial is the Humax F1 satellite STB, which is reasonably priced at about £70. Besides the cost criteria, the STB also allows manual configuration of the reception signal transmission parameters [77]. Such capability is an important factor as this allows the STB to lock on to DVB-S signals modulated at different frequencies, symbol rates and FEC rates to recover the original data stream.

This satellite STB uses the Philips Semiconductor TDA8044A SDD chip. The chip communicates with the rest of the system using an inter-IC bus, I^2C-bus. Figure 2-11 shows a typical satellite STB design, where modules in the front-end and decoding-end of system communicate with each other using the I^2C-bus. Various controls and status registers can be accessed via the I^2C-bus [78]. The chip also conforms to the DVB-SPI for data interfacing and the de-modulated MPEG-2 TS can be extracted from the STB by performing slight modifications. The STB has previously been successfully modified to perform on-site data logging [59]. In this research, this hardware is used as a demodulation hardware and verification tool. It is also used as a verification tool by accessing the status register of the STB using in-house I^2C compatible data acquisition module (DAM) that taps into the I^2C-bus of the satellite STB [59].

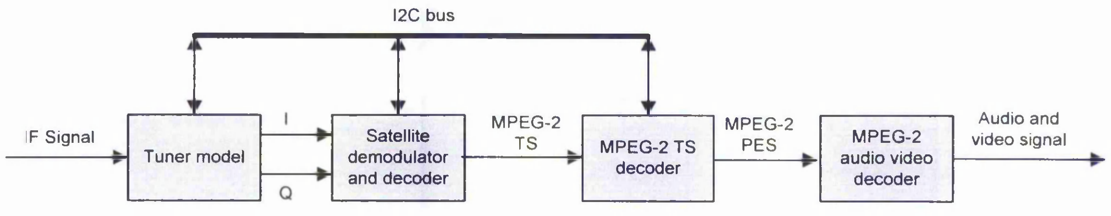


Figure 2-11: Satellite STB architecture block diagram

2.9.1 I^2C-bus

The I^2C-bus is a simple bi-directional 2-wire bus inter-IC control developed by Philips [79]. The two bus lines are made up of serial data line (SDA) and serial clock line (SCL). It is developed to provide simple operation with extremely broad range of I^2C compatible devices from Philips and other suppliers. One of such device is the Philips TDA8044A DVB-S decoder chip mentioned. As a multi-master bus, more than one device that is capable of initiating a data transfer can be connected to the bus. Connection to each device is software addressable, via the bus, by a unique address using a standard I^2C protocol.

When transmitting or receiving on an I^2C bus, a device that initiates the data transfer is a master. It also generates the clock signals to enable the transfer. At the same time, the device that is addressed is a slave. With both lines, SDA and SCL, maintaining at HIGH state when the bus is free, a transition of HIGH to LOW on the SDA line while SCL is HIGH signals a START condition, while a LOW to HIGH transition on the SDA line while SCL is HIGH indicates a STOP condition. The START and STOP conditions are always generated by the master device to initiate and terminate a transaction on the I^2C bus. Every byte on the SDA line is 8-bit long and each byte must be followed by an acknowledgement bit. During acknowledgement, the SDA line is submitted to the receiver device. At this condition, the receiver holds the line at LOW state while a HIGH period for SCL is generated by the master device. The first byte after START condition is used to address the device. 7-bit addressing is used, while the least significant bit (LSB) is used to indicate READ or WRITE operation with HIGH or LOW state respectively. Figure 2-12 depicts the access protocol on the I^2C bus.

Write data

S	State Address	/W	A	Data	A	Data	A	P

Read data

S	State Address	R	A	Data	A	Data	/A	P

Last data byte

S	= Start condition
P	= Stop condition
R	= Read
/W	= Write
A	= Acknowledgement
/A	= Not acknowledged

Figure 2-12: I^2C bus communication protocol

3 Design and Implementation of DVB-S Channel Encoder

Using a top-down design approach, the DVB-S channel encoder is broken into smaller groups of designs based on their functions. This approach provides a clear definition of the overall system and how information flows between the processes. Generally, the encoder system performs four main functions, that is energy dispersal, outer coding, interleaving and inner coding, as described in section 2.8.2. Design requirements of the processes within these functions are examined prior to implementation.

The process of 'zero-padding' frame-conditioning was added in the encoder design to insert 16 bytes of separation in between the 188-byte MPEG-2 transport stream. This is to enable concatenation of the Reed-Solomon codes. As a result of the additional null-bytes inserted, processes such as the randomizer are slightly modified from the conventional design to accommodate the changes and still conforming to the DVB-S standard. Each function is also designed to be bypassable to allow each function to be capable of being individually enabled.

3.1 'Zero-padding' Frame Conditioning

The Reed-Solomon encoder, RS(204,188), error protected packets offers recovery of the original code words should an error of up to 8 bytes occur during transmission. However, the concatenation of 16 bytes of RS code at the end of every 188th byte of every transport packet changes the packet structures. The basic MPEG-2 transport multiplex packet stream of 188 bytes requires 16 bytes to be added to form a 204-byte stream. Such occurrence suggests that pre-conditioning of transport frames is required to allow the application of the RS encoder.

A simple experiment that involved simulation of Altera's RS encoder IP core was performed. In the experiment, two types of data stream were processed by the RS encoder and its outputs were examined. One of the data stream resembles a 188-byte MPEG-2 transport stream, while the other represents a stream of conditioned 204-byte transport packets. For both data streams, 47_{HEX} with PSYNC were used for synchronisation, signalling the beginning of a new packet.

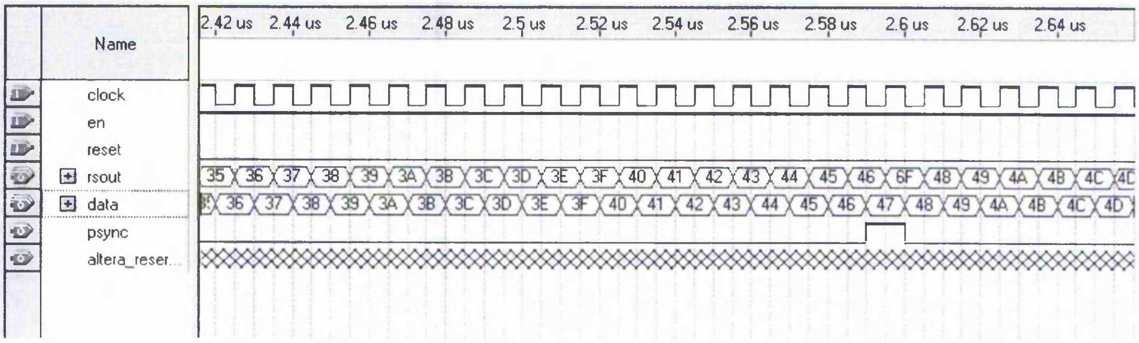

Name
2.42 us
2.44 us
2.46 us
2.48 us
2.5 us
2.52 us
2.54 us
2.56 us
2.58 us
2.6 us
2.62 us
2.64 us
clock
en
reset
rsout
data
psync
altera_reser...
35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44 45 46 6F 48 49 4A 4B 4C 4D
36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D

Figure 3-1: RS encoder output with 188-byte MPEG-2 transport packets

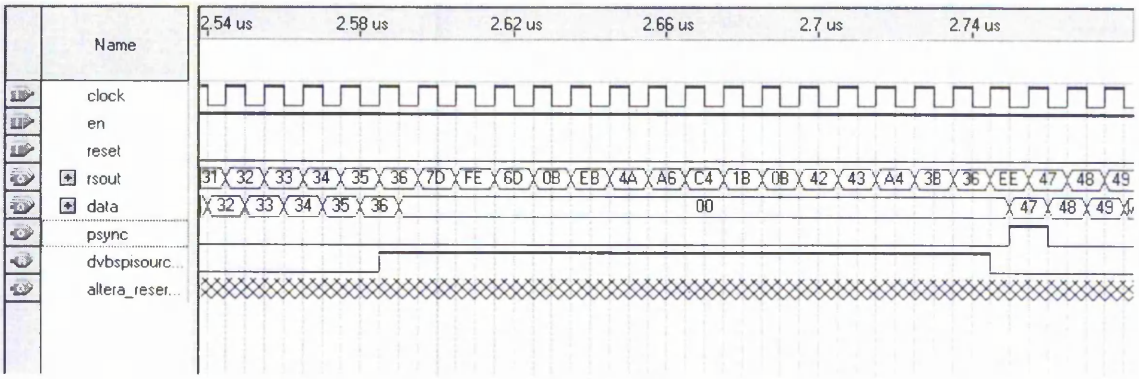


Figure 3-2: RS encoder output with 204-byte frame conditioned transport packets

The RS encoder failed to concatenate the 16 bytes of RS code to the packets when processing a standard MPEG-2 data stream. As depicted in Figure 3-1, a second transport packet carrying the PSYNC synchronisation bit has signalled the start of the packet before the RS codes of the previous packet were completely delivered. However, the following simulation shows a MPEG-2 data stream with 16 null-bytes between the end and the start of the next packet. A full set of RS codes can be generated before the start of the next packet, as shown in Figure 3-2. These experimental results have therefore confirmed that pre-conditioning of an MPEG-2 transport stream is required to conform to the use of RS(204,188) encoder.

Equation [3-1] defines the relationship between the interface rate and symbol rate for a generic DVB-S channel encoder [80]. The value for RS-rate for external MPEG framing with 188-byte frames indicated a change of data rate with the adoption of RS(204,188).

$$\begin{aligned}
\mathit{Symbol\ rate} = {} & \mathit{Interface\ bit\ rate} \\
& \times \mathit{Framing\ overhead} \\
& \times \frac{1}{\mathit{RS\text{ - }rate}} \\
& \times \frac{1}{\mathit{FEC\text{ - }rate}} \\
& \times \mathit{Modulation\ factor}
\end{aligned} \qquad \textbf{[3-1]}$$

$$\begin{aligned}
& \mathit{where,} \\
& \mathit{Framing\ overhead} = 1 \quad \mathit{for\ external\ MPEG\ framing} \\
& \mathit{RS\text{ - }rate} = \frac{188}{204} \ \mathit{for\ external\ MPEG\ framing\ with\ 188\ byte\ frames} \\
& \mathit{FEC\text{ - }rate} = \frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{5}{6} \ \mathit{or} \ \frac{7}{8} \\
& \mathit{Modulation\ Index} = \frac{1}{2} \ \mathit{for\ QPSK}
\end{aligned}$$

The process of frame conditioning is confirmed to involve changing the data rate of the transport stream, where the output data rate would have to be theoretically more than 204/188 (1.085) times faster than the input data rate. This would then enable 16 null bytes to be inserted at the end of each packet. As the transport packets are streamed back-to-back against one another, the packets will have to be paused while the insertion occurs. Such operation requires a First-In-First-Out (FIFO) memory function, where data can be temporarily written and read, in the sequence of it being written, when required. As the rate of which the data is written differs from the rate of reading, a dual clock FIFO is required.

Altera Dual Clock FIFO Megafunction

The Altera dual clock FIFO megafunction [81] is provided with the Altera Quartus II software. It is capable of using two independent clocks for writing and reading. This megafunction can be parameterised to implement any width or depth combination, where the only limitation is the available memory space in the device itself. To help eliminate the effects of metastability on this complex megafunction, six status signals were supplied to indicate empty, full or the number of words stored at both write and read side of the FIFO. The metastable state is a quasi stable state where the output of the FIFO is unpredictable. It usually occurs when the data input violates the setup or hold time and is marginally triggered [82].

In this design, the dual clock megafunction was implemented to have memory capacity of 8 bits x 256 words. As the depth of the FIFO must be a power of two, 256 words of memory capacity would easily buffer one 188-byte MPEG-2 packet. The status signal `rdusedw` was configured to show the amount of data that was ready to be read. With the system requiring an output data rate that is faster than the input data rate, it is more important to be monitoring the words that are ready to be read. The megafunction is configured in Legacy mode, which means that the requested data would be made available on the first clock cycle after a read request, `rdreq`, is asserted [81].

3.1.1 Dual Clock FIFO Controller

The dual clock FIFO controller is one of the most important modules designed in this research. It manages the operations of the FIFO that enables major encoding processes of the channel encoder, such as the Reed-Solomon encoding. Running at different clock frequencies, the dual clock FIFO was controlled using two different control algorithms, one to manage data writing into the FIFO and the other to manage data read from the FIFO.

FIFO Write Control

The FIFO write clock, `wrclk`, was made synchronous with the interface clock of the system as data packets are streamed directly into its data input ports, `data[7..0]`. As shown in Figure 3-3, data is buffered into the FIFO on the first rising clock edge after write request, `wrreq`, is asserted. At power up, the FIFO write control mechanism would remain in reset status, `IN_RESET`. During this state, a simple reset sequence is performed to the FIFO; `wrreq` is set high for a period of one clock cycle, to ensure that its registers' values are set to 00_{HEX}.

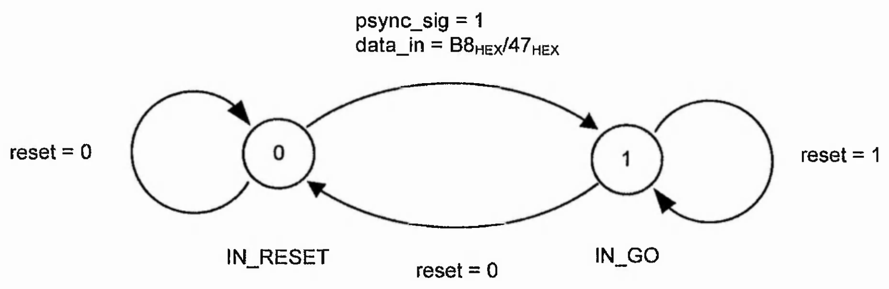


Figure 3-3: State diagram for FIFO write control

While remaining at `IN_RESET` state, the FIFO write control would monitor for the MPEG-2 synchronisation (SYNC) byte, which is 47_{HEX}, accompanied by a PSYNC bit. These conditions would trigger a state change from `IN_RESET` to `IN_GO`. At the same time, the transition would signal the FIFO to start buffering data streams by setting the `wrreq` to high. Valid MPEG-2 packets, signalled 'high' on the data-valid (DVALID) line, would be captured into the buffer. This is done in compliance with DVB-SPI standard [70].

FIFO Read Control

The FIFO read clock is asynchronous with the interface clock. Theoretically, the data from the buffer should be read 204/188 times faster than it being written into. As with `wrreq`, the requested data is made available on the first rising clock edge after `rdreq` is raised to high [81]. Remaining at reset status, `CTRL_RESET`, from power up, the FIFO read control would monitor the number of words buffered in the FIFO. The registers on this side of the FIFO were also reset by the sequence identical to the one described in FIFO write control. To avoid the issue of metastability, `rdusedw` is chosen in the design as it shows the number of words ready to be read from the FIFO. As soon as 188 bytes of a packet are readily available, the state machine would trigger a change of state from `CTRL_RESET` to `CTRL_READ`. At the same time, the change of state also changes the signal for `rdreq` to start reading from the FIFO.

A counter tracking the number of words read from the FIFO would prompt for a change of state from `CTRL_READ` to `CTRL_STOP` when the 188th word is being read. `rdreq` would be set to low and the counter reset. At this stage,16 null bytes would be introduced to the output. Just as the 16th null byte is clocked, the `CTRL_STOP` state would change back to `CTRL_READ`, indicating `rdreq` to start reading the next packet. As the read control mechanism monitors the `rdusedw` for the amount of words available in the FIFO, `CTRL_STOP` state would change to `CTRL_FILL` state if the content of the FIFO is near empty. The output stream would then be filled-in with a null packet while the content of the FIFO is being replenished, as depicted in Figure 3-4. Complete block diagram and VHDL source code listing are presented in Appendix III.

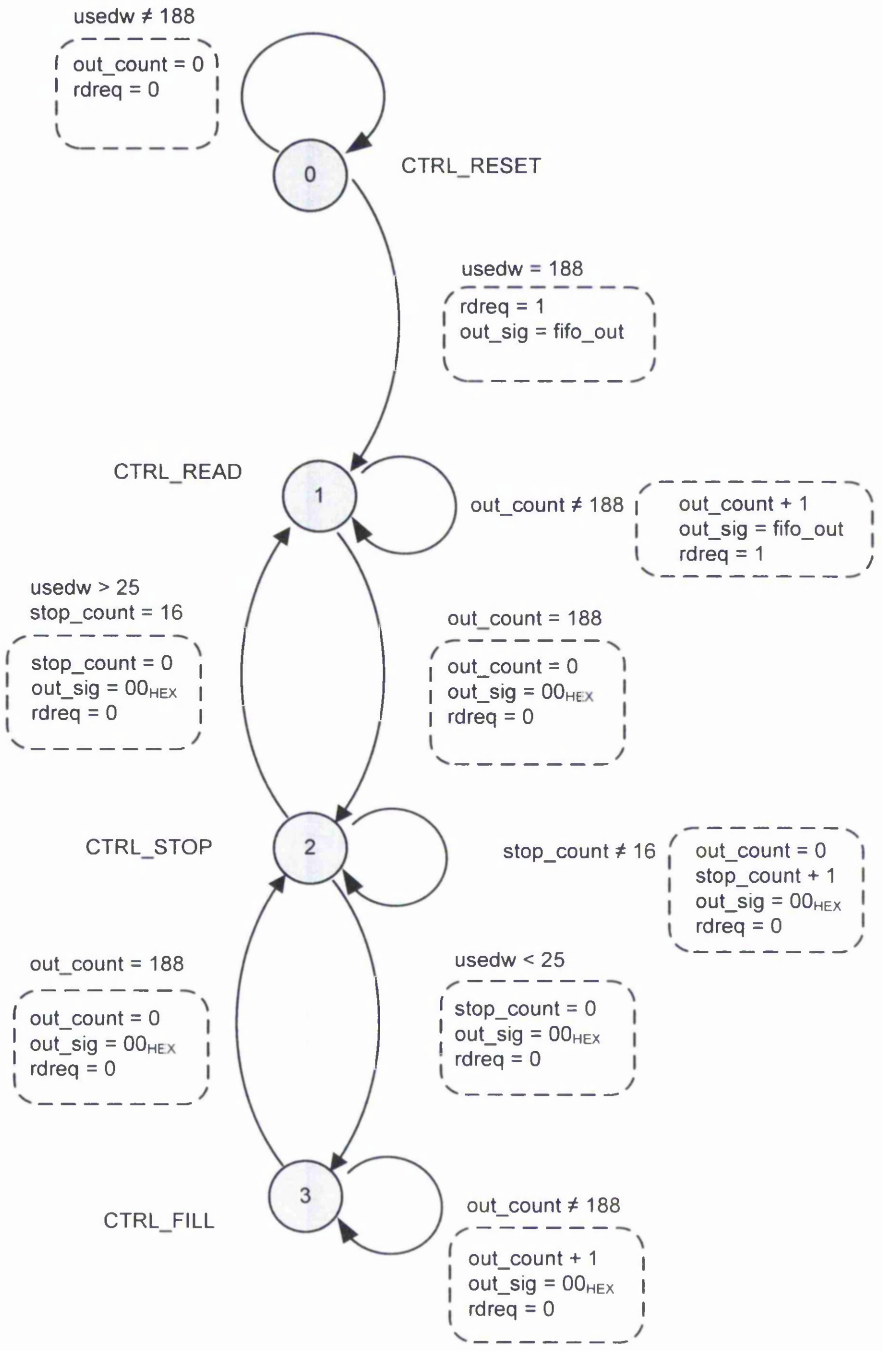


Figure 3-4: State diagram of FIFO read control

3.2 System Clock Management

In order to satisfy the clocking requirements of the system in general, and the functions of each module, the 20 MHz on board clock is supplied to a PLL and a clock divider modules to generate various clock frequencies. Figure 3-5 shows how the various clocks are distributed throughout the entire system. The 20 MHz clock is supplied to a PLL to generate a slower interface clock and the faster system clock. The clocks are then further divided down to supply the modules. Generally, all functional blocks of the DVB-S channel encoder are working on clocks derived from the faster clock.

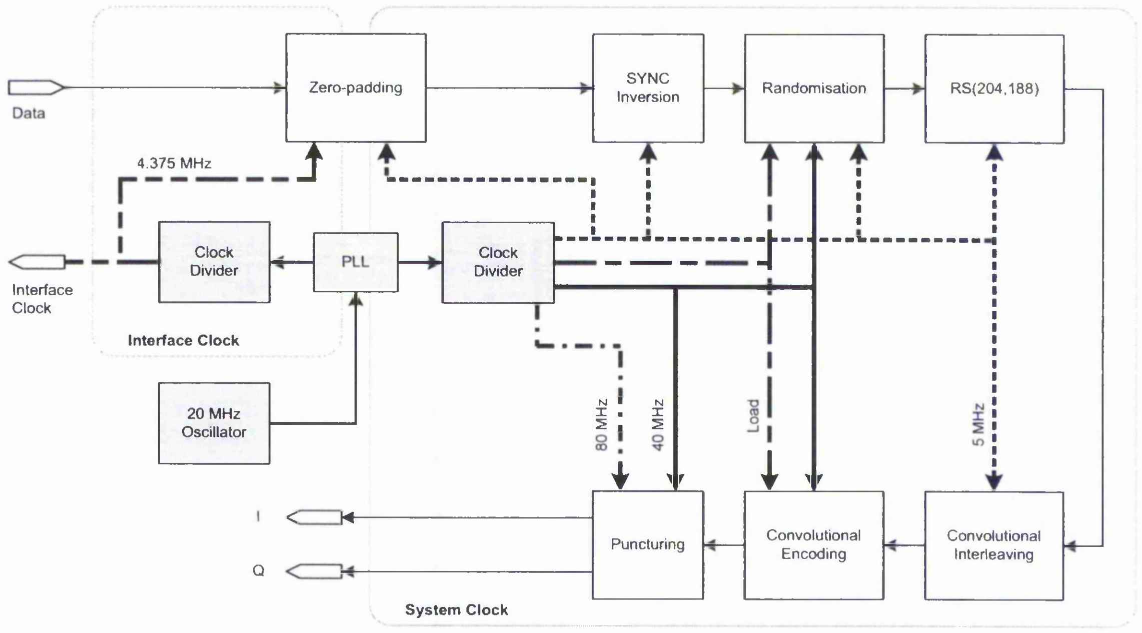


Figure 3-5: Clock distribution of the channel encoder system

3.2.1 Phase Locked Loop on Cyclone

Phase locked loops (PLLs) are used in designs to synchronise internal clocks with external clocks, run internal clocks at higher frequencies than external clocks, and to minimise clock delay and clock skew [54]. The Altera Cyclone allows implementation of up to two PLLs on an FPGA. The PLLs are enabled by a Megacore function, exactly as the other Megacores provided with Quartus II software. Likewise, the Megacore function is highly parameterized. The Cyclone PLLs are capable of performing multiplication, division and phase shifting. Both internal and external clock outputs are supported with programmable duty cycle. In the context of this research, the PLL is used to exploit the on-board 20 MHz clock. The 20 MHz clock can be multiplied to supply faster interface and system clocks that are required by the system.

Clock Multiplication and Division

Scaling factors are used to generate the output of the Cyclone PLLs. The relationship between the frequency of the voltage control oscillator (VCO), f_{vco}, the reference frequency, f_{ref}., pre-scale divider, N, feedback factor, M, input frequency, f_{in}, output frequency, f_{out}, and post-scale counters, $G0$ and $G1$, are given in Equations [3-2] [54]:

$$
\begin{aligned}
f_{ref} &= \frac{f_{in}}{N} \\
f_{vco} &= f_{ref} \times M = f_{in} \times \left(\frac{M}{N}\right) \\
f_{out} &= \frac{f_{vco}}{G0} = f_{in} \times \left(\frac{M}{N \times G0}\right)
\end{aligned}
\qquad \textbf{[3-2]}
$$

Each Cyclone PLL can support up to two outputs. When the output frequencies are different, the VCO frequency will be set to a value that is the least common multiple (LCM) of the VCO frequencies required by the output frequencies. Equations [3-2] can be used to evaluate the output frequencies that are required. The VCO frequency must also be within its operating range, which is from 500 MHz to 1000 MHz. Hence, the Cyclone PLL is not capable of generating outputs for combination of frequencies that are not within the limits of its VCO.

Although the difference between the output frequencies can be more than (204/188) 1.085 times apart, they are ideally 1.085 times apart or as near to it as possible. Efficiency of the dual clock FIFO is optimised when the clock frequencies are ideally set as underrun of the buffer is least to occur. A packet of null bytes is added to the transport stream every time re-buffering occurs, affecting overall transmission efficiency. In the design of this channel encoder, output frequencies of 80 MHz and 70 MHz were generated by the Cyclone PLL, with clock ratios 4/1 and 7/2 respectively to the 20 MHz on-board clock. Although this combination of frequencies is 1.143 times apart, which is 5.3% outside of the theoretical value, the values are within the operating limits of the PLL and therefore can be generated.

3.2.2 Clock Divider Module

A clock divider circuit was implemented to produce clock rates that are suited for the individual modules. As the basic clocks generated by the PLLs are the fastest clocks in the system, the following clock specifications are required,

- 1/2 of System Clock (40 MHz) is required for randomizer, convolutional encoder, and puncturing as the load signal is required for 2-bit Parallel-to Serial (P2S) conversion.
- 1/8 of System Clock (5 MHz) is required for most functional blocks processing in bytes, including reading from the dual clock buffer, SYNC inversion, randomization, RS(204,188), Convolutional interleaving and convolutional encoding.
- 1/8 of Interface Clock (4.375 MHz) is required for the writing into the dual clock buffer in bytes.
- Load signal is required for 8-bit P2S converter in randomization process and convolutional encoding.

```
PROCESS (load_clk, load_reset)
        VARIABLE clk_count     : INTEGER RANGE 0 TO 7;
BEGIN
        IF (load_reset = '0') THEN
                clk_count := 0;
        ELSIF (load_clk'EVENT AND load_clk = '1') THEN
                clk_count := clk_count + 1;
                IF (clk_count = 3) THEN
                        load_sig <= '1';
                ELSE
                        load_sig <= '0';
                END IF;
        END IF;

        IF (load_reset = '0') THEN
                load_out <= '0';
        ELSIF (load_clk'EVENT AND load_clk = '0') THEN
                load_out <= load_sig;
        END IF;
END PROCESS;
```

Listing 3-1: Generation of load signal

Implementation of the clock divider module involved a counter from the Library of Parameterised Modules (LPM). The LPM provides a library of logic functions that are parameterised. As these modules are architecture-independent, the LPM modules are supported by various vendors [83]. Just as a Megafunction, the LPM is enabled and parameterised using a dialog window in Quartus II software. After customising its features, such as setting the number of bits and counting up or down, the module for the counter is generated. In this design, two 5-bit counters were generated, one for dividing System Clock and the other for Interface Clock. Outputs form these counters were connected to corresponding output pins representing different clock speeds. Output of the least significant bit (LSB) of the counter is 1/2 of the input clock, while, output of the next pin is 1/4, 1/8 and so on. As for the load signal, a counter was defined in VHDL as a reference and a HIGH state was sent to the output was after every eight clock pulses. The source code for generation of the load signal is presented in Listing 3-1.

3.3 Transport Multiplex Adaptation

The transport multiplex adaptation is also a pre-conditioning process that involves inverting the first MPEG-2 SYNC byte of every eight packets, from 47_{HEX} to $B8_{HEX}$. The process is also known as SYNC inversion. It is used for synchronisation purposes to scramble and de-scramble at transmitting and receiving ends of the DVB-S system respectively.

Each of the input data and its PSYNC state were sampled and checked to identify the MPEG-2 synchronisation byte, 47_{HEX}. The MPEG-2 SYNC byte will occur when a 47_{HEX} word is detected with a HIGH state for PSYNC signal. When this occurs for the first time, SYNC byte 47_{HEX}, `01000111`, would be inverted bit-wise to $B8_{HEX}$, `10111000`. After the SYNC byte, the other data bytes will be ignored and no changes will be done until the next SYNC byte. The following SYNC bytes were counted, but ignored as they would not be inverted until after the seventh SYNC byte.

3.4 Randomization

Randomization is used to comply with International Telecommunication Union (ITU) Radio Regulations and to ensure adequate binary transitions. It is used not only to prevent strings of all 0s and all 1s, but also short repetitive sequences. A typical randomizer operates bit wise. Input data bits are shifted in stages while feedback taps are taken and exclusive-ORed [84]. The result of this exclusive-OR is then exclusive-ORed with the input to generate the randomized output. The design of this functional module is greatly aided by a schematic diagram of the randomizer presented in [56], as shown in Figure 3-6. Nevertheless, an 8-bit parallel-to-serial (P2S) converter, a serial-to-parallel (S2P) converter and a control module are required as part of the system.

3.4.1 Randomizer Control Module

The randomizer control module was designed to ensure that data was randomized conforming to the standard. Randomization would only be applied to the first MSB of the first byte following the inverted MPEG-2 SYNC byte. The standard prescribes that PRBS generation shall continue during the MPEG-2 SYNC bytes of the subsequent seven transport packets, but with its output disabled to leave those bytes unrandomized. However, with the 'zero-padding' pre-conditioned packets, a new control was added to retain transparency. The PRBS generation was paused, after the last bit of the 188th byte, for a period of 16 bytes. This algorithm not only ignored the added 16 null-bytes but also maintained the period of PRBS at 1503 bytes. Figure 3-7 shows the generation of PRBS with the pre-conditioned packets. The randomization process was paused, indicated by 'P', to ensure that the additional null-bytes are transparent and hence conforming to the DVB-S standard.

3.4.2 8-bit Parallel-to-Serial and Serial-to-Parallel Converters

The reason a P2S converter is required in the design is due to the fact that the randomiser proposed in EN 300 421 processes the packets in bits. As the standard specified that the PRBS generator shall be applied from the most significant bit (MSB) first, the P2S converter was designed to ensure that data bytes are converted conforming to that requirement. The randomized single bit output was then re-converted back to parallel byte stream. Conforming to the standard, the bits were converted to byte with the first bit as the MSB.

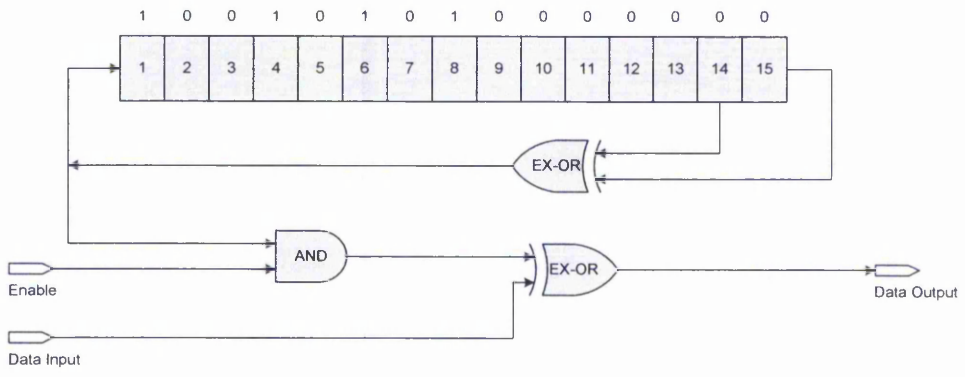


Figure 3-6: Randomiser schematic diagram

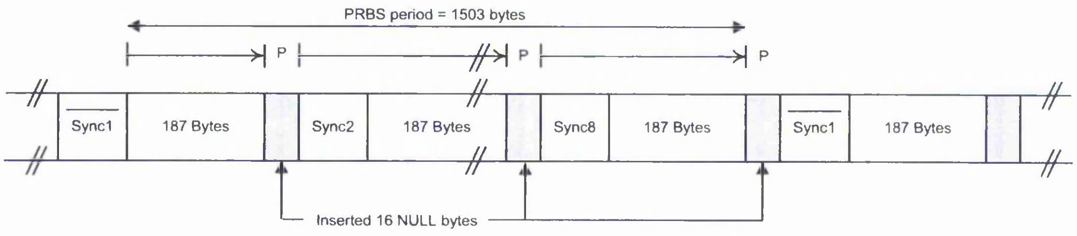


Figure 3-7: PRBS generation

3.5 Reed-Solomon (204,188) Encoder

Reed-Solomon codes are used for forward error correction. The Reed-Solomon encoder is capable of correcting burst errors in extreme-sensitivity compressed data streams, such as in DVB MPEG-2 transmissions. In the case of RS (204, 188), 16 bytes parity symbols are being concatenated at the end of each packet, providing capability of correcting up to 8 bytes of errors.

Initially, all registers in the encoder are set to zero and the switch is set to stream out the input data. The input data are also fed into the encoder one-by-one byte until the end of the frame. During this process, the input data that go into to encoder is multiplied with the coefficients of the generator polynomial, g_0 to g_{15}, as shown in Figure 3-8 [85]. The products are then added as they are being loaded into the registers.

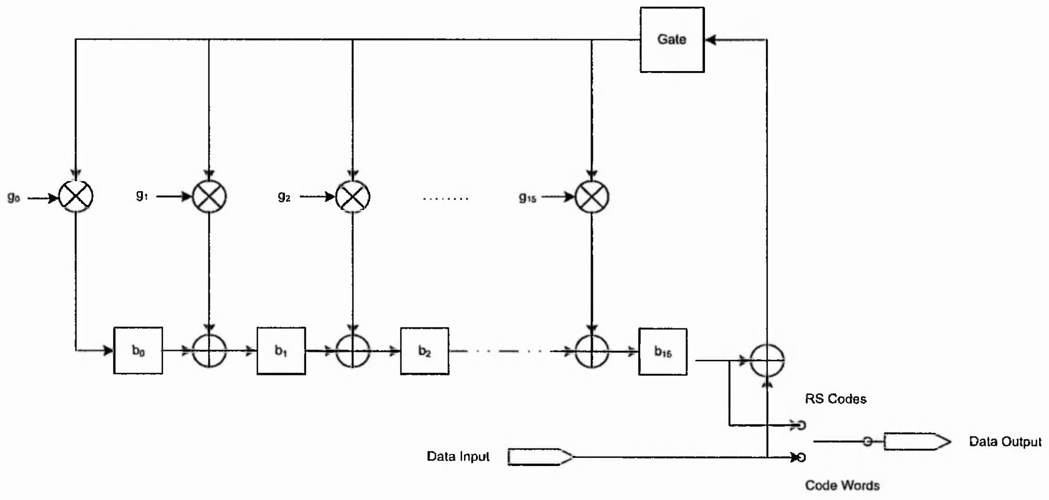


Figure 3-8: Functional diagram of Reed-Solomon (204,188)

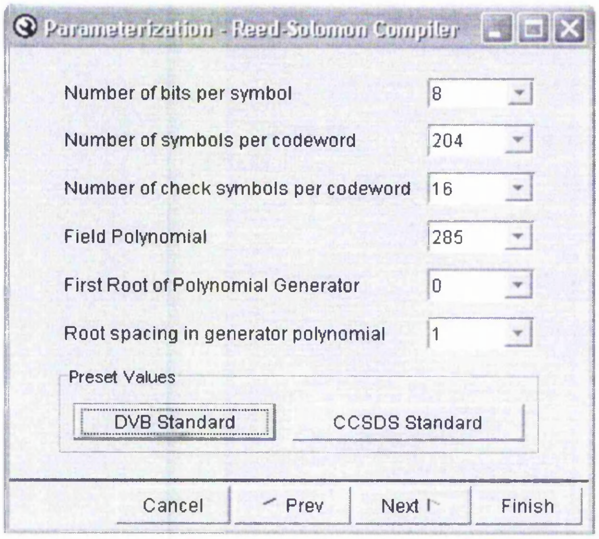
After the last bit of the last data byte is received, the switch is toggled to begin transmitting the computed Reed-Solomon parity symbols. Simultaneously, the computation circuits are being cleared by inserting bytes of zeros.

3.5.1 Altera Reed-Solomon Encoder IP Megafunction

The Reed-Solomon encoder megafunction, IP-RSENC, is produced by Altera Corporation. It provides full support for Cyclone family FPGAs and the most of the other Altera devices. The megafunction is fully parameterised with preset values that ensures compliance with DVB standard.

As explained in section 2.3.1, this IP core has Altera's free OpenCore Plus evaluation features, that allows not only behavioural simulation of the Megacore function on the system, but also allows programming of the Megacore to a device for verification within a period of time. With a 20 MHz system clock, the megacore is expected to be able to run for 13500 seconds before it expires.

Newer versions of the Megacore package included an RS Compiler IP Toolbench. It provides a one-stop toolbar within the Quartus II software to view documentation, specify parameters and generate all files necessary for the Megacore. A full functional description and timing diagrams of the RS encoder Megacore are presented in the user guides where, in newer documentations, the issue of frame conditioning is addressed [86]. There must be some space between the end and the beginning of the next packet for the check symbols. Similar to the other Megacores, setting up the parameters of the RS encoder is done by means of a guided dialog box. In the case of a DVB standard RS(204,188), the module can be generated with the preset values, as shown in Figure 3-9.


Parameterization - Reed-Solomon Compiler

Number of bits per symbol	8
Number of symbols per codeword	204
Number of check symbols per codeword	16
Field Polynomial	285
First Root of Polynomial Generator	0
Root spacing in generator polynomial	1

Figure 3-9: RS encoder Megacore parameterization dialog box

3.6 Convolutional Interleaver

The convolutional interleaver is used to distribute errors that occur in bursts more evenly between the packets. It rearranges the sequence of the symbols in a pattern that can be inversely rearranged at the de-interleaver to restore the sequence. Usually used with the RS encoder, the convolutional interleaver can enhance the protection against longer burst errors. As the RS(204,188) encoder can recover up to 8 bytes of errors per packet, this interleaving scheme can handle burst errors of up to 8x12=96 bytes or 384 QPSK symbols [87].

The operation of a convolutional interleaver is straightforward. Based on Forney's proposal, each word of the MPEG-2 TS is cyclically connected to the input of a branch of the interleaver [88]. In turns, the branches input a word while shifting out the oldest word, as illustrated in Figure 3-10. The input and output switches are synchronised. According to the DVB-S standard, the convolutional interleaver may be composed of *I=12* branches, with depth of *Mj*, where *M=17* and *j=branch index* [56].

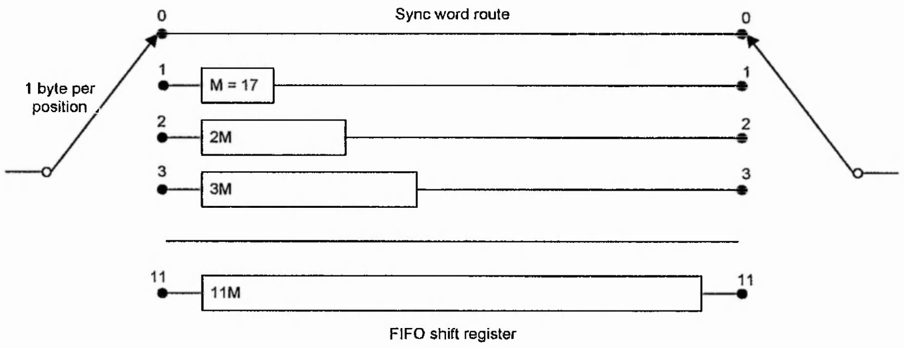


Figure 3-10: Functional diagram of convolutional interleaver

3.6.1 Interleaver Control

The interleaver control module can be represented by the input and output selector pins pictured in Figure 3-10. At every clock cycle, the selectors would synchronously connect to one branch following the sequence. Simultaneously, as one byte of data is sent into the buffer on the branch, one byte of data is being read out. Functioning as the selector pins, the design of this control mechanism involves sending and retrieving one byte of data on one branch at a time.

As illustrated in Figure 3-11, 11 states were assigned on the state diagram. This is to allow one branch of the convolutional interleaver to be represented as one state. Although this technique seems inefficient, it simplifies its functions and implementation. The design would change a state for every clock cycle. Therefore, one byte of data would be sent and retrieved from the branch.

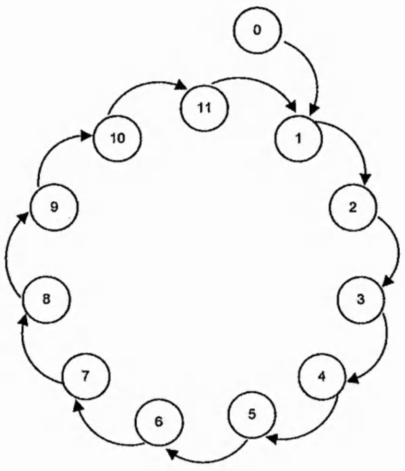


Figure 3-11: State diagram of interleaver control

3.6.2 FIFO Control

The control module is designed to manage the inputs and outputs of a synchronous buffer. A general algorithm is devised for the first FIFO, that is branch index, $j=1$, as depicted in Figure 3-12. The algorithm is reused to control the rest of the FIFOs by changing the number of clock cycles the control module remained in the state of ENABLE.

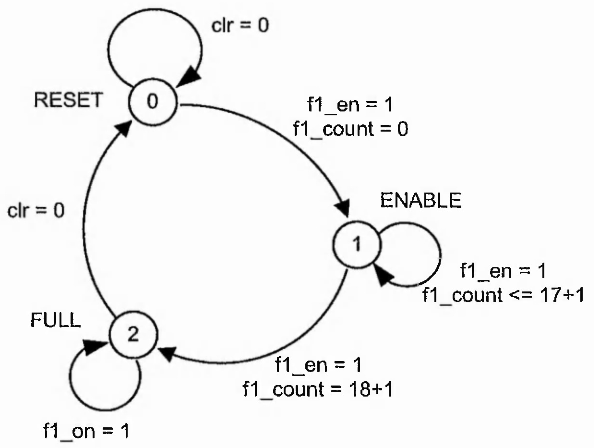


Figure 3-12: Generic state diagram of FIFO control

This general algorithm allows the FIFOs implemented as the branches of the interleaver to behave like a byte-wise shift register. At the start, the algorithm waits for the enable signal from the interleaver control module. During the ENABLE state, data is input and stored into the FIFO. Based on the depth of the each branch, *Mj*, the state is changed to FULL, when the FIFO is filled to required depth. During the FULL state data can be written into and read from the FIFO. At this stage, the FIFO synchronously inputs a new word while the oldest word stored is shifted out at the output and hence, this is the operation of a FIFO shift register. Listing 3-2 presents the VHDL source code for branch index 1 of the FIFO shift register controller. A slight change was made to the binary value of `fx_usedw` to correspond to shift register depth of the other branch index.

```
PROCESS (f1_clr, f1_en, f1_usedw)
BEGIN
        IF (f1_clr = '0') THEN
                f1_aclr <= '1';
                f1_rdreq <= '0';
                f1_wrreq <= '0';
        ELSIF (CONV_STD_LOGIC_VECTOR(f1_usedw,5) >= "10000") THEN
                f1_rdreq <= f1_en;
                f1_wrreq <= f1_en;
                f1_aclr <= '0';
        ELSE
                f1_wrreq <= f1_en;
                f1_rdreq <= '0';
                f1_aclr <= '0';
        END IF;
END PROCESS;
```

Listing 3-2: FIFO shift register controller for branch index 1

3.6.3 Single Clock FIFO Megafunction

As with the dual clock FIFO megafunction, the Altera single clock FIFO megafunction [81] is also provided with the Altera Quartus II software. The writing and reading operations are synchronous. Similarly, this megafunction can be parameterised to implement a variety of width or depth combination; the only limitation is the available memory space in the device itself. Such flexibility is useful as the single clock FIFO is used in this design to implement the 11 branches of the interleaver with different sizes. As described in [56], the branches are FIFOs with depth Mj cells. Table 3-1 shows the actual size of the buffers on each of the branches and the corresponding size of the FIFO used.

j	*Mj*	*FIFO size*
1	$1\times17=17$	32
2	$2\times17=34$	64
3	$3\times17=51$	64
4	$4\times17=68$	128
5	$5\times17=85$	128
6	$6\times17=102$	128
7	$7\times17=119$	128
8	$8\times17=136$	256
9	$9\times17=153$	256
10	$10\times17=170$	256
11	$11\times17=187$	256

Table 3-1: Required depth for FIFO

3.7 Convolutional Encoder

The convolutional encoder processes data bits serially and continuously. It is a type of discrete convolutional code generator using modulo-2 arithmetic [89]. Known for its high redundancy encoding, convolutional coding provides high rates of error correction, yet is simple to design and requires minimum circuitry [84].

As introduced in section 2.8.6, the DVB-S standard used a rate 1/2 convolutional code with constraint length, $K=7$ [56]. This means that the convolutional encoder consists of a 7-stage shift register. With rate 1/2, there would be two modulo-2 outputs for each information symbol [90]. The operations of the modulo-2 adders are defined by the binary equivalent of polynomials given in [56]:

$$X = g_1 = 171_{\mathrm{OCT}} = 1111001$$
$$Y = g_2 = 133_{\mathrm{OCT}} = 1011011$$

With reference to the 7-bit binary equivalent values of the polynomials, the shift registers are connected to the modulo-2 adders where the bits are represented by '1' and no connections for bits that are represented by '0'.

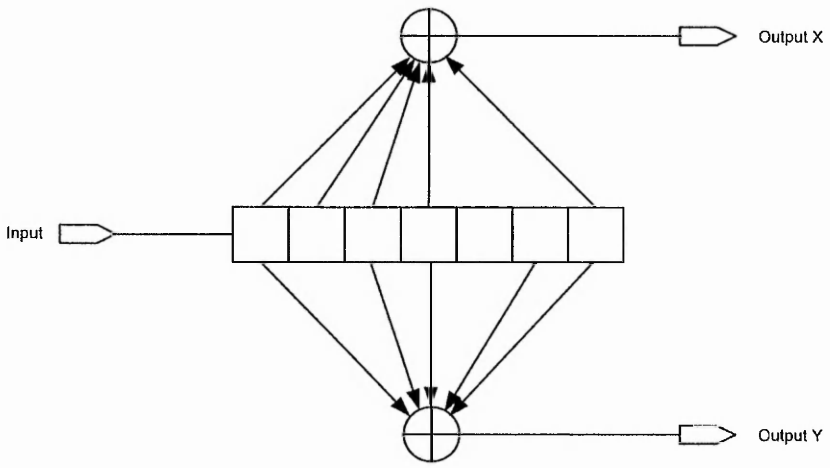


Figure 3-13: Functional diagram of convolutional encoder

As the convolutional encoder processes in continuous serial bits, an 8-bit P2S converter is required. The 8-bit P2S converter module coded as part of the randomizer was re-used in the design of this convolutional encoder. The coded module is placed in the functional block library and implemented as part of the design.

```
PROCESS (cnv_clk, cnv_clr)
BEGIN
        IF (cnv_clr = '0') THEN
                in_latch <= "0000000";
        ELSE
                IF (cnv_clk'EVENT AND cnv_clk = '1') THEN
                        in_latch(6) <= cnv_in;
                        in_latch(5 downto 0) <= in_latch (6 downto 1);
                        x_out <= in_latch(6) XOR in_latch(5) XOR in_latch(4) XOR
                                  in_latch(3) XOR in_latch(0);
                        y_out <= in_latch(6) XOR in_latch(4) XOR in_latch(3) XOR
                                  in_latch(1) XOR in_latch(0);
                END IF;
        END IF;
END PROCESS;
```

Listing 3-3: Convolutional Encoder

Figure 3-13 shows the simple design of the rate 1/2 convolutional encoder. Implementation of the design was coded using VHDL, where the input signals are shifted in a 7-bit bus signal to represent the workings of a shift register, as presented as Listing 3-3. The modulo-2 adders were represented by the Exclusive-OR operation [85]. The signals were described using the XOR operator already defined in the VHDL library.

3.8 Puncturing

The added redundancy due to convolutional coding has reduced the code rate by 1/2. This means that every bit data is being represented by 2 data bits. The efficiency of convolutional encoding can be increased by not transmitting every bit of data across the medium. This process is called puncturing. Based on a pattern standardised for DVB-S systems, some bits of the encoded data can be excluded from being transmitted. The puncturing process for DVB-S systems is capable of increasing up to 7/8 of the overall code rate. The process of puncturing must be applied conforming to the punctured code definition, where the puncturing patterns are specified.

	Code rates				
	1/2	*2/3*	*3/4*	*5/6*	*7/8*
Puncturing Code	*X: 1* *Y: 1*	*X: 10* *Y: 11*	*X: 101* *Y: 110*	*X: 10101* *Y: 11010*	*X: 1000101* *Y: 1111010*
Output	$I=X_1$ $Q=Y_1$	$I=X_1Y_2Y_3$ $Q=Y_1X_3Y_4$	$I=X_1Y_2$ $Q=Y_1X_3$	$I=X_1Y_2Y_4$ $Q=Y_1X_3X_5$	$I=X_1Y_2Y_4Y_6$ $Q=Y_1Y_3X_5X_7$

Table 3-2: Punctured code definition

Table 3-2 shows the puncturing patterns as defined by DVB for code rates 1/2, 2/3, 3/4, 5/6, and 7/8 [56]. The puncturing codes can be analogous to a mask that is filtering the data bits received from the convolutional encoder output, X and Y, where '1' represents transmitted bit, and '0' represents punctured bit. The bits that are being punctured are the ones that are not transmitted. The punctured code definition also describes the pattern which the punctured data bits are being sent at the output, I and Q.

3.8.1 Rate 3/4 Punctured Code

Rate 3/4 puncturing was chosen to be designed and implemented as it was already adopted by the NTU campus network trial system. Signal measurements and analysis of the campus trial system were based on rate 3/4 DVB-S encoding [59]. According to the punctured code definition, one of the methods to achieve the output pattern, I and Q, of rate 3/4 puncturing is by converting the two inputs into a one-bit data stream. As depicted in Figure 3-14, the inputs, X_n and Y_n, are interlaced to form a continuous bit stream. As with the inputs, the puncturing code is also interlaced to form a single bit masking pattern. When the rate 3/4 punctured code is applied to the stream, every third bit of the data stream will be deleted and therefore not transmitted. During this clock cycle, the two bits of data that were not deleted are clocked to generate the I and Q outputs that is compliant to the DVB-S standard.

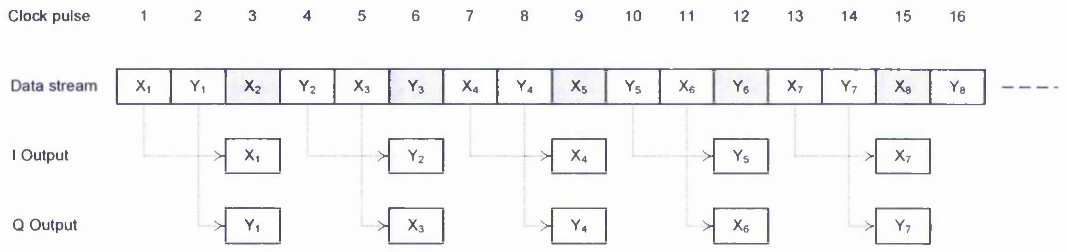


Figure 3-14: Rate 3/4 punctured code pattern

A 2-bit P2S converter was coded based on the source codes of the 8-bit P2S converter designed as part of the randomization module. At the very core of this rate 3/4 puncturing design is a modified S2P converter with a sampler. The sampler would operate with a controller that counts the clock pulse in a 3-bit cycle. At every third bit, a signal would be sent to the S2P converter, as shown in Figure 3-14. This signal would have a 37% duty ratio. When the signal goes HIGH, the values sampled in the S2P converter is then latched to the I and Q outputs. Designs for the puncturing module are presented in Appendix III.

3.9 Configurability

With the aim of realising a hardware system that is reconfigurable to allow changeable coding schemes to be tested over the university campus network trial, each module in the coding process of the DVB-S channel encoder is modified to allow itself to be switched on or off. As shown in Figure 2-8, the DVB-S coding algorithms are performed in a sequence, where energy dispersal is performed first, followed by outer coding, interleaving and finally inner coding. This means that when the modules are integrated as a system, data must be passed between the modules in the way defined by [56].

To facilitate the flexibility of disabling a particular coding module, each module is designed with a bypass channel that is controlled by an enable input-pin. To alternate between 1/2 rate and 3/4 rate punctured convolutional encoding, a simple switching design was employed to connect the I and Q outputs with data from the convolutional encoder for 1/2 rate coding and the puncturing module for 3/4 rate coding.

3.9.1 Module Bypass

The module bypass design is applied at the inputs and outputs of a module. The inputs and outputs are connected to avoid the data from being coded by the module, when required. The decision to bypass is controlled by changing the value of the enable input-pin, `en`, as depicted in Figure 3-15, where X represents the coding modules of the system.

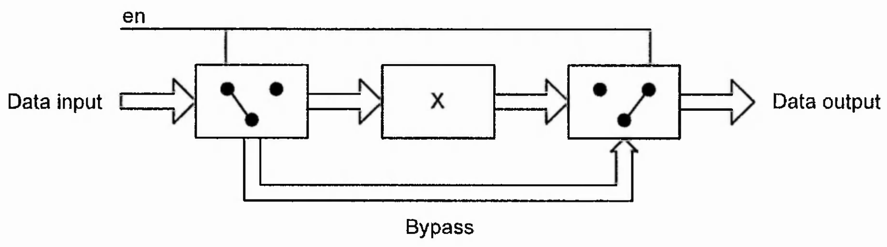


Figure 3-15: Bypass enabling design

This design was applied to modules performing SYNC inversion, RS(204,188) encoder, and convolutional interleaver. Listing 3-4 presents the source code of the module bypass design written and implemented as part of the RS(204,188) encoder to enable the module to be switched on or off. As for the randomizer, the enable pin was already a standard feature of the randomizer module. As shown in Figure 3-6, the enable pin was connected to allow disabling of the randomization process on the data.

```
PROCESS (en_clk, en_in)
BEGIN
        IF (en_clk'EVENT AND en_clk = '1') THEN
                IF (en_in = '0') THEN
                        en_dataout <= en_datain;
                        en_syncout <= en_syncin;
                        en_dataenin <= "00000000";
                        en_syncenin <= '0';
                ELSE
                        en_dataout <= en_dataenout;
                        en_syncout <= en_syncenout;
                        en_dataenin <= en_datain;
                        en_syncenin <= en_syncin;
                END IF;
        END IF;
END PROCESS;
```

Listing 3-4: Module Bypass

4 Verification of DVB-S Channel Encoder

Verification of the designed modules was performed individually using HDL software simulation. This ensures that the modules are functionally verified prior to system integration. A data source that emulated the MPEG-2 TS was specially coded for this purpose. Simulation results obtained were analysed and correlated against output of simulation tools used in the industry, such as Simulink.

Hardware verification was also performed on a systemic level. This completed system is prototyped on the FPGA development board and tested as part of the 42 GHz MWS campus network trial. Hardware verification methods also included accessing status register of an STB as well as the use of 'live' digital TV signals. A PCB was also designed to ensure maximum I/Q power transmission and signal integrity during interfacing with external devices.

4.1 MPEG-2 Transport Stream Emulation

MPEG-2 Transport Stream (MPEG-2 TS) emulation is the most important entity in the verification processes of the DVB-S channel encoder. A single continuous stream of MPEG-2 packets is required as an input at almost every stage of the simulation. As the system is designed to conform to the DVB Synchronous Parallel Interface (DVB-SPI) [70], the emulated MPEG-2 TS is synchronised with a supplied clock, and accompanied with a PSYNC signal to flag at the start of a packet and DVALID to signal for a valid packet.

The design for MPEG-2 TS emulation involved an 8-bit count-up counter, enabled from the LPM [83], and a control mechanism. The counter output is used here to generate the emulated MPEG-2 data stream. To conform to MPEG-2 framing standard, 47_{HEX} and a PSYNC signal are used at the start of a packet. This is achieved by referring the output of the counter; as the counter counts up, the control module outputs a PSYNC when it reaches 47_{HEX}. The control module then resets the counter when it reaches $6C_{HEX}$ to restart counting from 00_{HEX} on to 46_{HEX}. This count cycle would generate a 188-byte packet.

Figure 4-1 shows a section of the output of the MPEG-2 TS emulator. The data output, `data`, is a stream of changing code words as a result of the count-up counter output. The 188-byte packet ended at the code word 46_{HEX} and the start of the following packet continued on with 47_{HEX} accompanying a PSYNC signal at its output, `psync`.

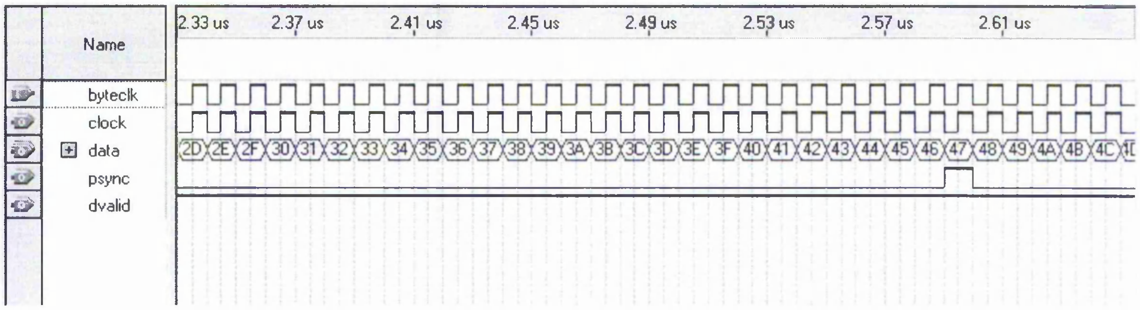

Name
2.33 us
2.37 us
2.41 us
2.45 us
2.49 us
2.53 us
2.57 us
2.61 us
byteclk
clock
data
psync
dvalid
2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C

Figure 4-1: 188 byte MPEG-2 transport stream emulation

When required, this emulator design can be altered to generate 188+16-byte transport packets. Instead of $6C_{HEX}$, the counter is reset when it counts to CC_{HEX} to generate 204 bytes of data. The DVALID signal is switched off for the last 16 bytes of the packet as they are not valid RS codes concatenated to the packet [70]. The 16 extra bytes added to packets are output as null bytes, 00_{HEX}.

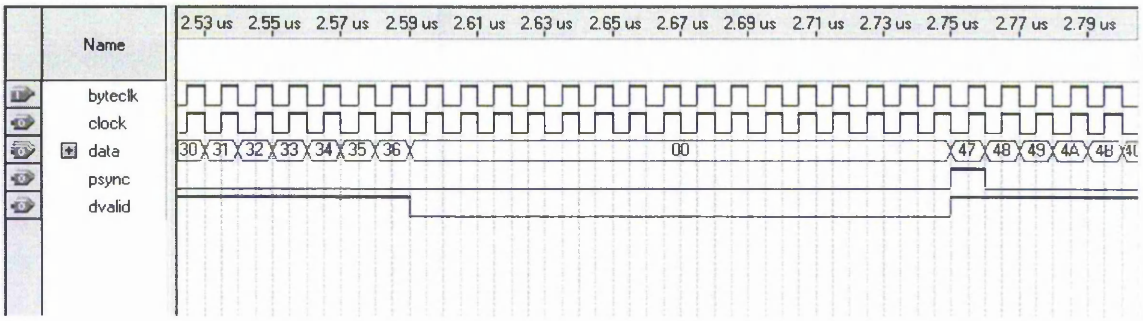


Figure 4-2: 188+16 byte transport stream emulation

A section of the simulation output of the MPEG-2 TS emulator is presented in Figure 4-2. It shows the end of a packet with the 16 null-bytes added. As described, DVALID output, `dvalid`, is switched off for the additional null-bytes. A new packet then follows on.

4.2 'Zero-padding' Frame Conditioning

As previously described in section 3.1, in order to allow application of RS coding, frame conditioning is employed to the MPEG-2 transport multiplex. Before coding processes begin, the 188-byte MPEG-2 TS have 16 null-bytes added at the end of every packet. The design of this functional block included a dual-clock FIFO with a controlling module.

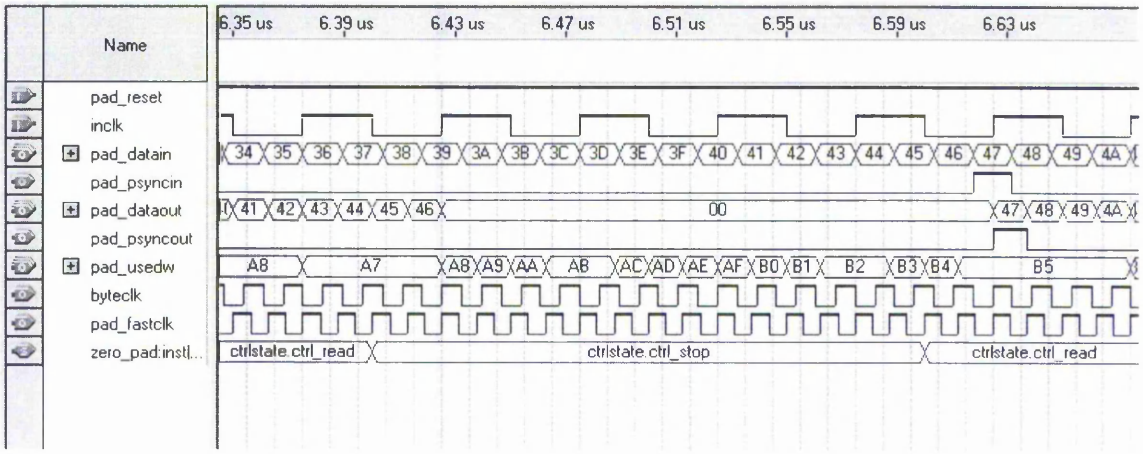


Figure 4-3: Null-bytes insertion for frame conditioning

During verification, the functional block uses the MPEG-2 TS emulation output as data source. A PLL was used to generate clock rates that are identical to the system's clock rates; `pad_slowclk` was connected to the 70 MHz clock while `pad_fastclk` was supplied with 80 MHz clock. Figure 4-3 shows a section of the simulation output of the frame conditioning process. Emulated MPEG-2 TS entered the module via input port, `pad_datain`, at its interface rate. Zero-padded transport packets are output via `pad_dataout` output port at a faster rate to accommodate the 16 additional null-bytes. PSYNC signals were generated to accompany the SYNC bytes to provide synchronisation of the TS.

As anticipated, the amount of data buffered in the FIFO decreases as the zero-padding process is performed. The gradual reduction of data in the FIFO is due to the difference between the interface clock and the system clock. Bigger difference from the theoretical 204/188 rate would result to faster reduction of the buffered data. However, as the amount of data buffered is almost empty, in which less than 25 bytes left, the `CTRL_STOP` state changes to `CTRL_FILL` to stop the output of data for as long as 188 byte to top up the amount of data stored. Figure 4-4 presents the inserted null-packet when the FIFO is being re-buffered. For the purpose of identification the null-packet was set to output BB_{HEX} words.

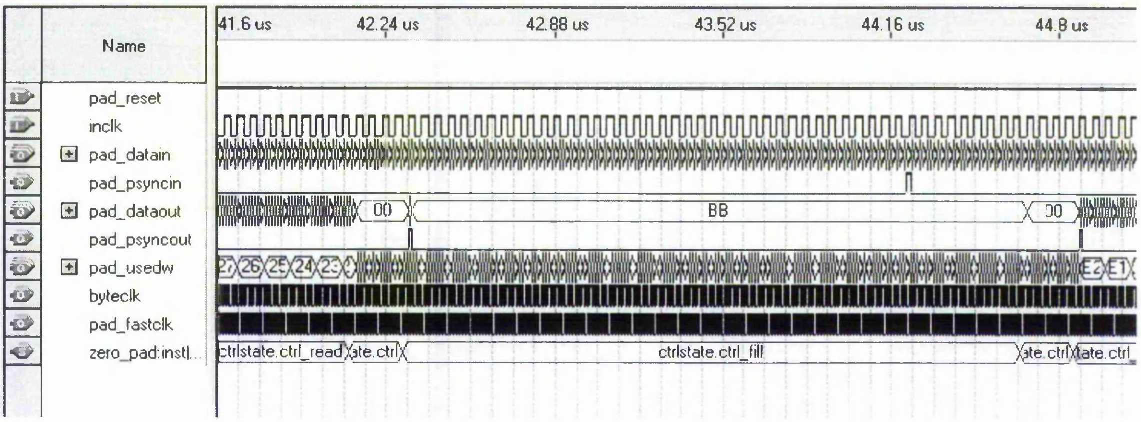


Figure 4-4: Null-packet insertion for buffering

4.3 Transport Multiplex Adaptation

The transport multiplex adaptation was designed to bit-wise invert MPEG-2 SYNC bytes from 47_{HEX} to $B8_{HEX}$ for synchronisation. Only the first of every eight SYNC bytes are inverted. During simulations, the MPEG-2 TS emulation module can be used to provide MPEG-2 packets into the SYNC inverter module. However, the simulation can be simplified by shortening the input packets into the test module. Complete MPEG-2 packets are not required to verify the SYNC inversion process that occurred once in every eight packets. Therefore, as shown in Figure 4-5, a simulation of the SYNC inversion module was performed using 4-bytes per packet stream.

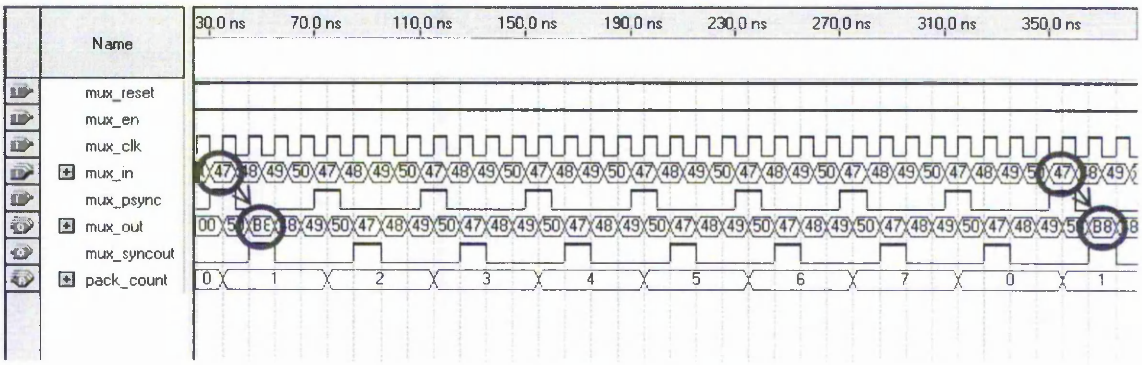


Figure 4-5: Simulation of SYNC inversion

A continuous stream of packets entered the module at `mux_in` input port, where the SYNC bytes were identified by the accompanying PSYNC signals. At the output, `mux_out`, the first SYNC byte was inverted and the number of packets was counted, `pack_count`. After the 8th packet, the following SYNC byte was again inverted, as indicated on the figure.

4.4 Randomization

The randomization process is used in the transmission to facilitate energy dispersal in compliance with radio regulations. Details and design of this functional block were described in section 3.4. Although the DVB-S [56] publication has provided schematics of the PRBS generator, designs were implemented for serial-parallel conversions and a specially devised control mechanism.

Verifications on the functionality of the control mechanism are important to ensure that changes to the frame conditioned transport stream are transparent to the randomization process. With the added null-bytes, PRBS generation is required to be temporarily paused, to avoid scrambling of the null-bytes, and continued from the following SYNC byte. Scrambling of the additional data bytes would cause corruption of the entire transmission as the PRBS period would run for more than the specified 1504 bytes [56].

The data source used for simulation of the randomizer module is the emulated MPEG-2 TS. The SYNC inversion module was used on the TS prior to randomization. The data source was then supplied to the randomization module and was first converted to a serial bit stream. Output of the parallel-to-serial conversion can be probed at `P_Sout`. Figure 4-6 presents a snapshot of the simulation of the randomizer module, where the PRBS generator is paused for the null-bytes. The signal `bitclk_ctrl` was coded at the control instructions as an enable switch for the PRBS generator. Therefore, the signal was disabled for the entire duration of the additional null-bytes.

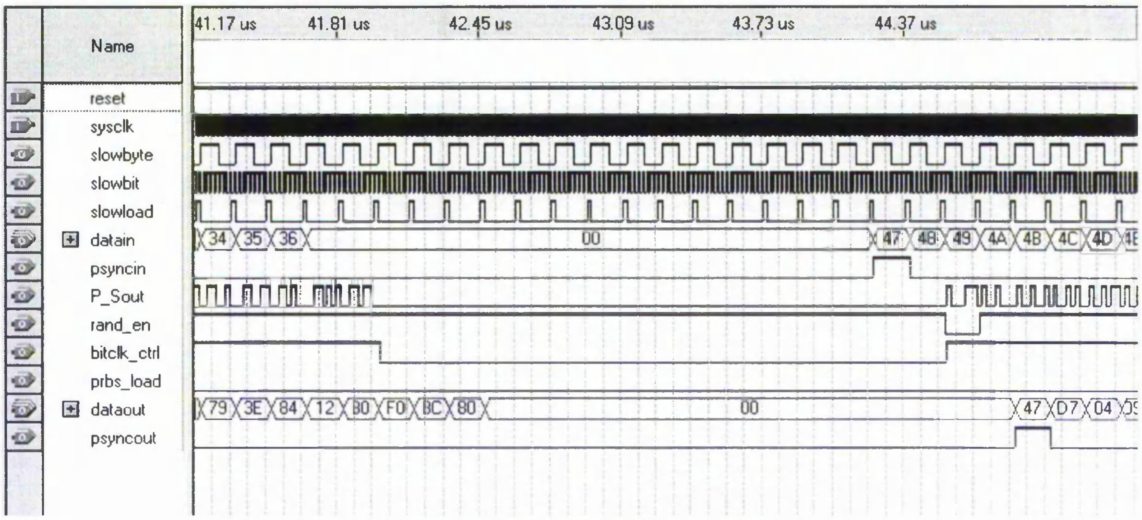


Figure 4-6: PRBS paused at zero-padding

The general functions of the control mechanism were also verified by ensuring that the `prbs_load` signal is initiated when an inverted SYNC byte occurs. This signal would reset the PRBS generator to have its initial sequence "`100101010000000`" loaded. Another signal verified during simulations was the enable signal for randomization, `rand_en`. This signal is used to ensure that the MPEG-2 SYNC byte of the TS remained unrandomized, although the PRBS generation continued [56]. Therefore, as observed in Figure 4-6, the `rand_en` signal was disabled as the serialised SYNC byte "`01000111`" entered the PRBS generator and remained unrandomized.

4.5 Altera Reed-Solomon (204,188) Encoder IP

The Reed-Solomon (204,188) encoder was implemented using Altera's megafunction IP core. It adds 16 extra redundancy bytes at the end of each 188-byte MPEG-2 packets to provide error recovery of up to 8 bytes long during transmission. Several simulations were performed prior to using this IP core to investigate its input requirements. One such simulation involved inputs from the MPEG-2 TS emulator which were set to generate 188-byte MPEG-2 packets with 16 additional null-bytes. As shown in Figure 3-2, full 16-byte RS codes were successfully concatenated at the end of each transmission packet, when the null-bytes were included as part of the input transport stream.

4.6 Convolutional Interleaver

The convolutional interleaver is made up of an array of FIFO shift registers with different capacities. With a control algorithm, data was distributed over 12 branches of FIFO shift registers to increase the efficiency of the RS(204,188) encoder. A longer burst of error in transmission can be sustained by the RS encoder. A description of the design of the convolutional interleaver module can be found in section 3.6.

Verification of the implemented module was performed with the use of Altera's convolutional interleaver megafunction. Functional simulation output of the implemented module was correlated against output of the IP core using the same set of input data. The module and the IP core were configured to connect to the emulated MPEG-2 TS generator, as illustrated in Figure 4-7. The functionality of an IP core made for commercial licensing is guaranteed as its developer must ensure that the IP core was built in compliance with industrial standards. An IP core can be simulated and its output is reliable unless the core in incorrectly parameterised.

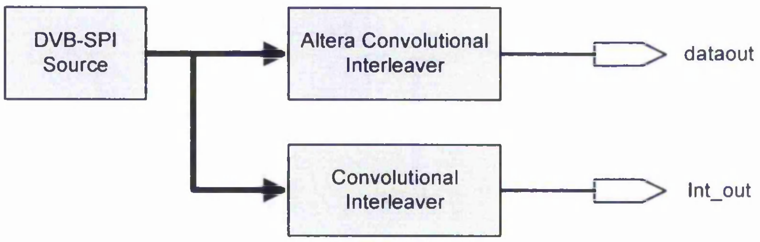


Figure 4-7: Convolutional interleaver verification configuration

A section of the simulation output of the convolutional interleaver is presented in Figure 4-8. Simulation output `dataout` is the output of the IP core, while output port `int_out` is the output of the module implemented in the system. The modules were connected to the same emulated MPEG-2 TS used for other tests. This simulation successfully validated functionality of the implemented convolutional interleaver module as the output values of `int_out` matched the output value of `data_out`. However, the delay of one clock cycle was observed between the outputs. Optimisation of the commercial IP megafunction design has resulted in the timing difference against the output of the implemented module.

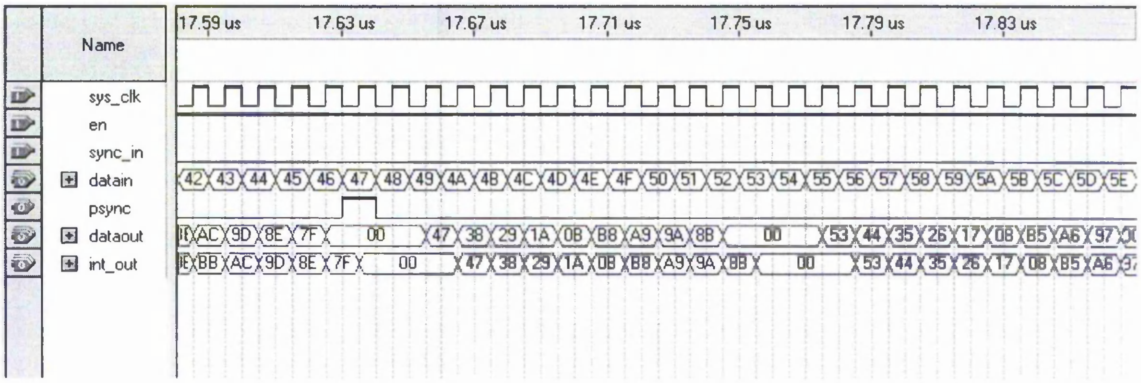


Figure 4-8: Convolutional interleaver verification using IP core

4.7 Punctured Convolutional Encoder

As the punctured convolutional encoder was implemented in two separate parts, namely convolutional encoder and puncturing modules, the verification results are also presented separately. Verification of these modules was carried out by correlation of VHDL simulation traces with results of Simulink simulations.

4.7.1 Simulink Simulations

MATLAB is a high-level technical computing language. It features an interactive environment for algorithm development and data analysis [91]. Simulink is an extension to MATLAB that allows block diagrams representing system models to be created and edited in a graphical environment. It provides a graphical user interface to accurately design, simulate, implement, and test control, signal processing, communications, and other time-varying systems [92]. Add ons, such as the communications blockset, extend the Simulink environment for simulation of the physical layer of communication systems and components.

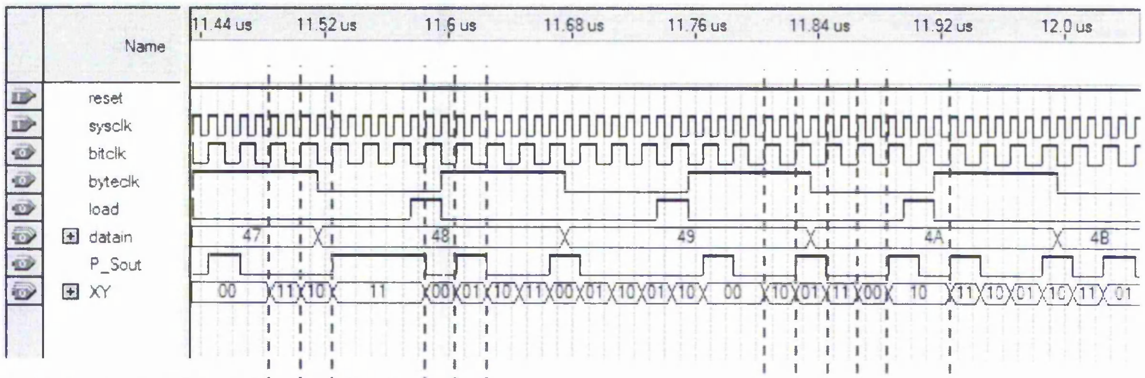
With the communications blockset installed, Simulink was used to simulate the convolutional encoder and the puncturing process. These processes were parameterised in the Simulink environment to comply with DVB-S standard published in [56]. With a known set of input data, the Simulink blocks were simulated and outputs were recorded. With Simulink being used in the industry to design and simulate commercial communication models, the Simulink component blocks are in compliance with industry standards to provide accurate representation of its functionalities. Therefore, verification of the convolutional encoder and puncturing module can be performed by correlating Simulink simulation results against simulation outputs of the VHDL coded modules, using the same set of input data.

4.7.2 Convolutional Encoder

The convolutional encoder is a simple channel coding technique that adds a large amount of redundancy for error correction [84]. Its simple design performs modulo-addition to the input data bits continuously based on polynomials described in section 3.7.

Although verification of this module can be performed manually using pen and paper, a Simulink simulation was used to correlate against the output of HDL simulation to ensure validity. HDL simulation output of the implemented convolutional encoder was obtained by input of a short sequence of incrementing data starting from 47_{HEX} to 52_{HEX} to emulate the start of a MPEG-2 packet, as presented in Figure 4-9(a). Similarly, the Simulink convolutional encoder block was simulated using identical input data, as presented in Figure 4-9(b).

(a) HDL Simulation Output



(b) Simulink Simulation Output

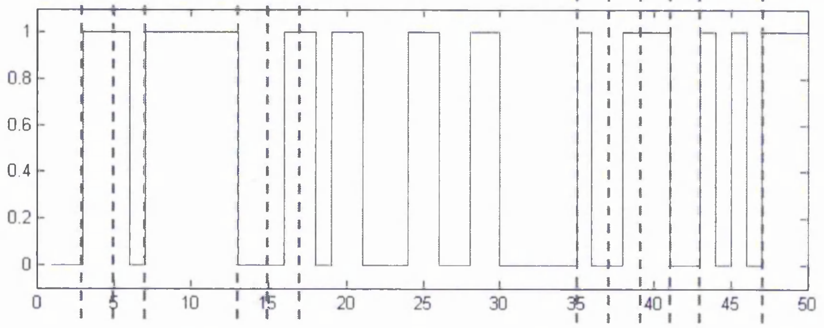


Figure 4-9: Simulation outputs of implemented convolutional encoder

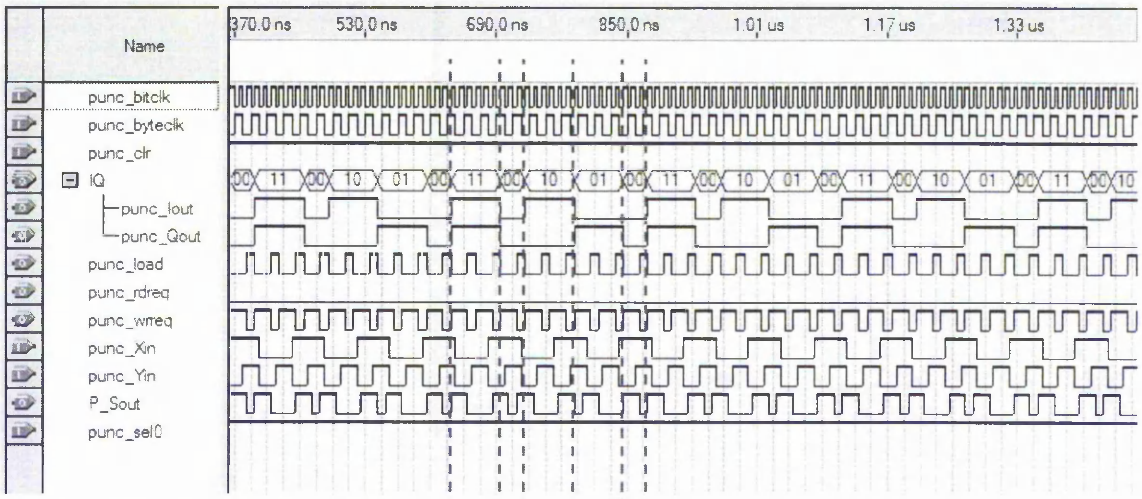
By associating the serial output, XY, on Figure 4-9(a) and the Simulink simulation output of a DVB-S standard convolutional encoder on Figure 4-9(b), the output of the designed convolutional encoder is exactly as the same as the output of a DVB-S standard convolutional encoder, as guided by the dotted lines, when the same input values are used. Therefore, this correlation process had successfully verified the functionality of the designed convolutional encoder.

4.7.3 Puncturing

The puncturing process involves deletion of data bits in a pattern that conforms to the DVB-S standard. Puncturing is performed to increase overall code rate of the system and increase FEC efficiency after the large amount of redundancy that was added into the encoding by the convolutional encoder. The punctured code definitions for all levels of efficiency are presented in section 3.8.

A rate 3/4 puncturing module was designed and developed as part of the DVB channel encoder. Verification on this module was performed by correlating the HDL simulation output against Simulink simulation output of a DVB-S standard puncturing. A two-bit counter was used to generate loops of incrementing binary values, 00_{BIN}, 01_{BIN}, 10_{BIN} and 11_{BIN}, as inputs to the puncturing module. Figure 4-10 shows the HDL simulation output of the designed puncturing module. Using identical input data, the Simulink puncturing block was also simulated and its simulation result is presented as Figure 4-10(b).

(a) HDL Simulation Output



(b) Simulink Simulation Output

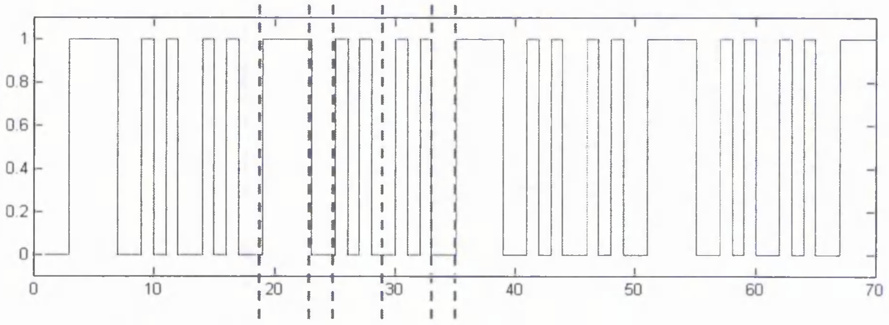


Figure 4-10: Simulation output of puncturing

As shown in Figure 4-10(a), serial output of the designed puncturing module, `IQ`, is used to correlate with the output of the Simulink simulation, Figure 4-10(b). As guided by the dotted-lines, the simulation output of the designed puncturing module is exactly the same as the simulation output of the DVB-S standard puncturing when the same input values are used. Therefore, the functionality of the designed puncturing module was successfully verified by this matching correlation.

4.8 Hardware Verification using Set-Top-Box

One verification process that can validate the operations of the implemented DVB-S channel encoder on a systemic level is by using a satellite STB. As the satellite STB is the receiving-end of the transmission link, it demodulates and decodes the received signal to remove the redundancy that was included during the encoding process conforming to DVB-S standard. The original stream of information is then recovered. A brief walkthrough of the decoding process is described in section 2.8.7.

The instant prototyping capability of FPGAs played an important role in this systemic hardware verification process. The designed channel encoder system can be programmed onto the FPGA on the spot for hardware testing. Important probe points and status flags of the designed system are described within the codes of the modules. By probing these points and the output of the module, functional faults and defects of the system can be quickly identified on a signal analyser.

Software simulation can be performed on a systemic level by providing a testbench to supply input stimulus to the design. In most cases of a VHDL based design, HDL testbenches are coded to monitor the characteristics of the signals within the design. As the Altera Quartus II software does not support HDL testbench, verification of the simulation output waveform alone is not feasible on a systemic level.

Verification using a STB involves programming the design onto an FPGA. A set of devices including the STB is required to be configured to facilitate the verification process. Sections 4.8.3 and 4.8.4 will describe the configurations and results of verification in entirety.

The first of two methods used to verify the design of the encoder system are by probing the output of the decoded signal. This method requires a known set of transport packets being encoded by the implemented DVB-S channel encoder (DVB-ENC) and sent to the STB. The decoded transport packets are then obtained from the STB and examined, where the DVB-ENC can only be proven to function correctly if the original and the decoded packets are identical. Typically, DVB-based channel decoder chip interfaces with an MPEG-2 decoder chip using DVB-SPI [93]. By tapping into the SPI, traces of the output of the channel encoder were obtained using a logic analyser.

The second method used is more technical. It is done by examining the status registers of the decoder chip in the STB. With the register indicating a lock on the demodulator, decoder clock and the rest of the FEC processes, the DVB-ENC is certain to be functioning correctly. As shown in Figure 2-11, modules in the front-end and decoding-end of a typical satellite STB system communicate with each other using a common inter-IC bus, the I^2C-bus. Various controls and status registers can be accessed via the I^2C-bus [94]. As the Humax F1-FOX satellite STB uses the Philips TDA8044A DVB-S decoder chip, which is an I^2C compliant device [95], the status register is accessed using a PC via a serial link to an in-house I^2C compatible data acquisition module (DAM) that taps into the I^2C-bus of the satellite STB [59].

4.8.1 I^2C-bus Data Acquisition Module

The DAM was originally designed to facilitate measurement of the 42 GHz wideband millimetre-wave signal using an inexpensive satellite STB [59]. The module can be attached to a Humax F1-FOX satellite STB to tap on its I^2C-bus. The module, reused in these verification processes, provides an interface with the I^2C-bus via a PC COM port. As mentioned, by having a means to access the I^2C-bus in the STB, the status registers of the Philips TDA8044A DVB-S decoder chip can confirm the functionality of DVB-ENC.

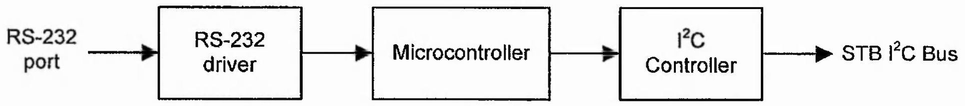


Figure 4-11: DAM architecture block diagram

The DAM architecture consists of an I^2C controller, RS-232 communication port, and a microcontroller. All tasks related to data acquisition are performed by the microcontroller. It acts as a transceiver that allows communication via a PC COM port in American Standard Code for Information Interchange (ASCII) format. With the I^2C controller, parallel communication can occur bi-directionally between microcontroller and I^2C bus. As explained, communication between with the I^2C bus is carried out on a byte-wise using polled handshake. The I^2C controller manages all the I^2C protocols to allow passive monitoring of the I^2C bus traffic on a PC. A utility shareware, Serial Device

Tester, is used to monitor serial communication at the PC COM port. Data in both ASCII and Hex format can be displayed simultaneously on a Graphical User Interface (GUI).

4.8.2 QPSK Modulation and Hardware Interfacing

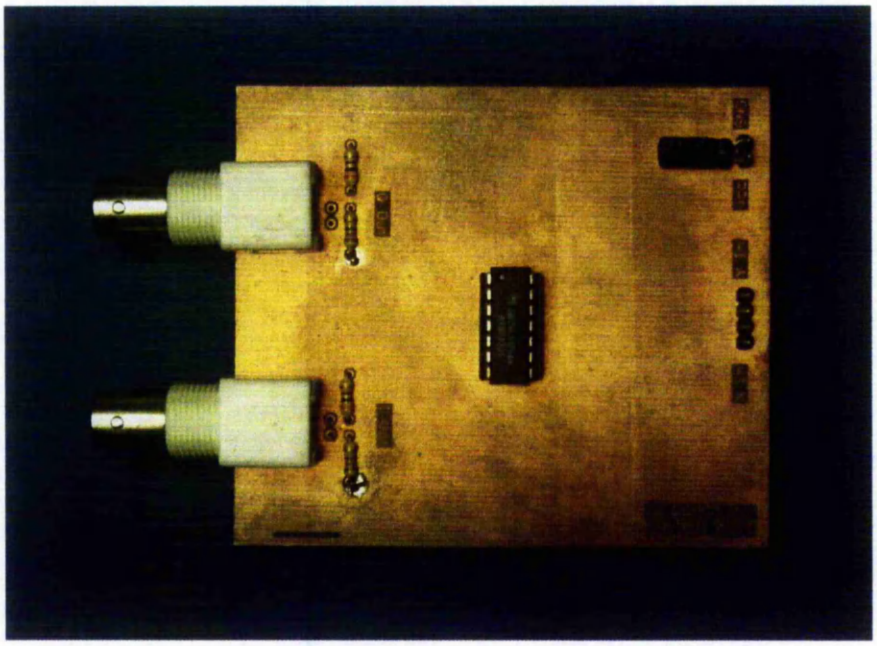
QPSK modulation is required at the I and Q outputs of the DVB-ENC to enable the use of a satellite STB. As an off-the-shelf consumer product, the STB is manufactured with a standard Satellite-Input. It is designed to receive signal that has been downconverted to an intermediate frequency (IF) signal by a low-noise block-downconverter (LNB) at the satellite-dish. The IF signal is then converted into an MPEG-2 TS by performing demodulation and forward error correction decoding as described in section 2.8.7.

During the verification process, QPSK modulation was performed using the Agilent Technologies ESG family signal generator to produce the IF signal that can be received by the STB. The signal generator was first used to condition the modulated signal. A negative offset of 25% were set to mimic a bipolar baseband signal for QPSK modulation. The signal is then modulated with carrier frequency of 1000 MHz and output at amplitude of -40 dBm.

Prior to using the signal generator, the equipment was fully investigated. The maximum input power into the device was found to be 1.0 Watt with input impedance of 50 Ω at each input. For external I and Q source, the recommended signal level is [96]:

$$\sqrt{I^2 + Q^2} = 0.5\, V_{rms} \qquad \textbf{[4-1]}$$

An I and Q output interface module was developed to match the input requirements of the signal generator. The module is positioned between the I and Q outputs of the DVB-ENC and inputs of the signal generator. As shown in Appendix I, a fully customised PCB was designed using Protel PCB design software. Several Gerber formatted files is then generated to plot and produce the PCB. The Gerber Format is also known as RS-274-D as the industry standard photo plotting language [97]. The interface module was designed to include a SN74LS126A bus buffers to reduce the power load on the I and Q output port of the FPGA. Using a simple parallel termination scheme, the line impedance was matched with a termination resistor of an equal value. Two 52 Ω resistors were used at each output as they were the nearest match available in-house. The scheme not only diminishes the first reflection, it also ensures maximum current loading at a HIGH output state [98]. To match match the power limitation at the input of the signal generator, a voltage divider circuit was used at both I and Q outputs.



Picture 4-1: I and Q Output interfacing module

With interface rate of more than 26 mega-symbols per second, the fast slew rate can contribute to noise generation, signal reflection, cross-talk and ground bounce. As mentioned earlier, a simple parallel termination scheme was used to avoid signal reflection. As shown in Picture 4-1, a 100 μF electrolytic capacitor was also placed adjacent to where the power supply enters the circuit board to filter low-frequency noise from the power supply [98]. To avoid occurrence of ground bounce and V_{CC} sag, surface mount decoupling capacitors of 0.01μF and 0.1 μF in parallel were also used.

4.8.3 Verification with Emulated MPEG-2 Transport Stream

Probing the output of the decoded signal is one of two methods mentioned earlier to verify the system using a STB. The configuration for this verification process can be separated into transmitting-end and receiving-end, as pictured in Figure 4-12. At the transmitting-end, the process involves insertion of a MPEG-2 TS into the DVB-ENC. Insertion of the MPEG-2 TS is done by re-using the MPEG-2 TS emulation module. The emulation module was applied and connected to the inputs and programmed on the FPGA as part of the hardware. The I and Q output signals of the DVB-ENC is QPSK modulated using the Agilent Technologies ESG signal generator. Configuration at the receiving-end includes STB_2, a TV to set-up the STB and a logic analyser to display traces of the decoded MPEG-2 TS. The probes were connected to the DVB-SPI data-bus of STB_2.

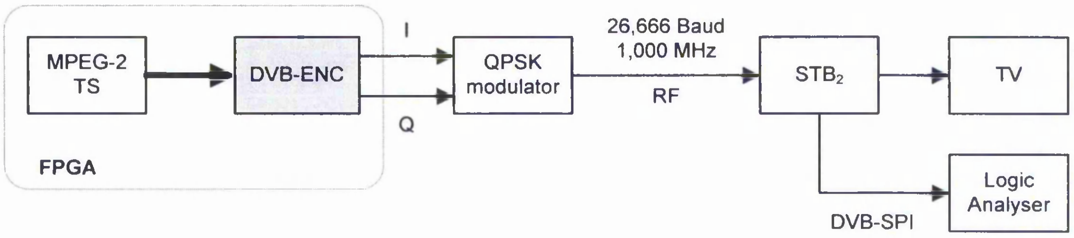


Figure 4-12: Configuration for verification with emulated MPEG-2 TS

Prior to running the verification process, STB_2 is required to have the L-Band frequency, symbol rate of the input signal and the FEC mode locked to the specification that is matching the transmitting-end. As the STB requires 'live' digital TV reception to lock and save a setting, a 'live' feed was re-modulated using Newtec NTC2063 professional DVB modulator. The re-modulated IF signal is then sent to tune STB_2. Table 4-1 shows matching the specification of the transmitting-end compared to the existing specifications. The verification process is then performed as depicted in Figure 4-12.

Specification	*Existing*	*Re-modulated*
L-Band Frequency	*1270 MHz*	*1000 MHz*
Symbol Rate	*17500 Baud*	*26666 Baud*
FEC Mode	*3/4*	*3/4*

Table 4-1: Specification difference between existing and re-modulated signal

This verification process has successfully proven the operations of the DVB-ENC with STB_2 effectively decoding the encoded source signal. The decoded data-bus traced at DVB-SPI of STB_2 was identical to the emulated MPEG-2 TS.

4.8.4 Verification with Digital TV Signal

As an extension to the previous verification test, a similar configuration of hardware can be used to verify the DVB-ENC using a 'live' digital TV multiplex. Similarly, the arrangement of hardware can be separated into transmitting-end and receiving-end. As shown in Figure 4-13, the I and Q output signals of the DVB-ENC is QPSK modulated prior to transmission. To facilitate the use of a 'live' digital TV multiplex, an IF signal from the university campus trial is demodulated and decoded by STB_1 to be re-modulated to match the requirements of the DVB-ENC. The 'live' multiplex stream is then connected into the DVB-ENC by tapping the DVB-SPI of STB_3, where the re-modulated signal was recovered.

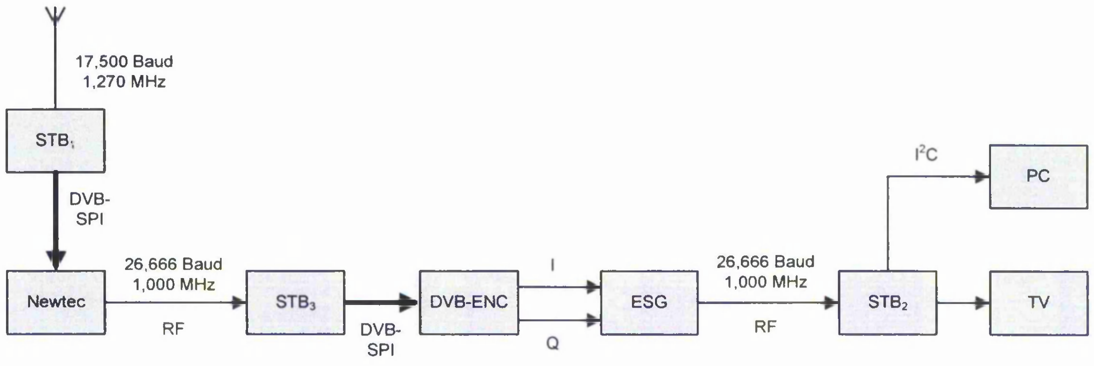
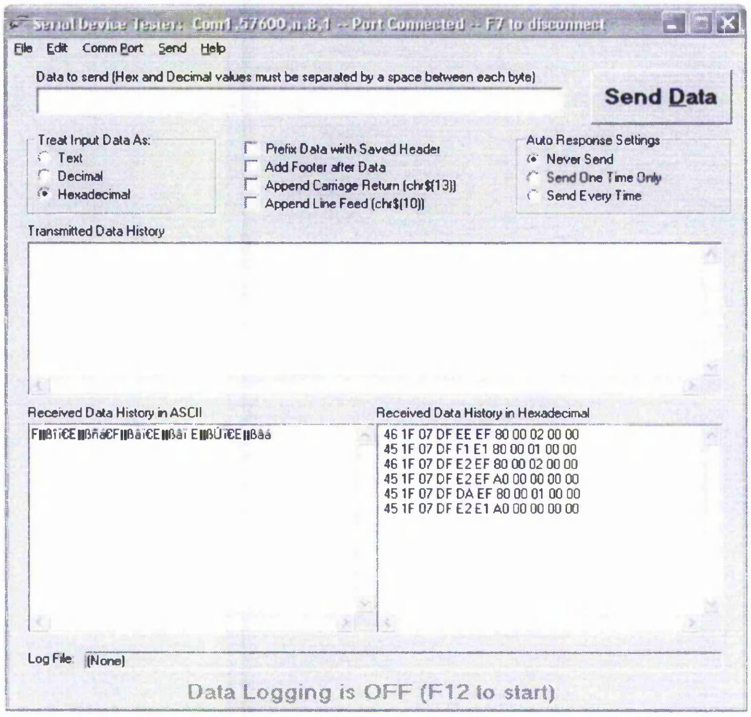


Figure 4-13: Configuration for verification with digital TV multiplex

At the receiving-end, the DAM is connected to STB_2 to allow monitoring of the status registers in the Philips TDA8044A DVB-S decoder chip at the I^2C-bus of the decoder system. A TV is used to set-up the STBs as well as displaying the decoded TV programme. At the same time, monitoring of the I^2C-bus is done on a PC using the serial COM port.

Accessing the PC serial COM port using Serial Device Tester software as a user interface, a total of 11 status-bytes were returned by the DAM. These data was identified as useful status flags and accessed via the I^2C-bus based on information and addresses published in [95]. One of the status addressed at 02_{HEX}, was identified as FEC locks status. This 8-bit flag registers a binary '1' to indicate different stages of the FEC are locked. Amongst the stages that can be monitored are Viterbi locked (FVL), de-interleaver locked (FDIL) and de-randomizer locked (FDRL) [95]. The output value of $1F_{HEX}$ is expected at this location under normal working conditions when all FEC stages are locked.

Figure 4-14 shows the output of 11 status flags accessed from the I^2C-bus via the DAM during the verification process. With $1F_{HEX}$ confirming that all the FEC stages are locked, the channel coding processes in the DVB-ENC were proven to conform to DVB-S standard and therefore demonstrated a successful development of the DVB-S channel encoder on an FPGA. Only one status register indicates FEC lock status, other status flags can be used to analyse characteristics of a channel. Estimated values for signal-to-noise ratio, automatic gain control, and number of channel bit errors can be obtained.


Serial Device Tester: Com1,57600,n,8,1 -- Port Connected -- F7 to disconnect
File Edit Comm Port Send Help
Send Data
Treat Input Data As:
Text
Decimal
Hexadecimal
Prefix Data with Saved Header
Add Footer after Data
Append Carriage Return (chr$(13))
Append Line Feed (chr$(10))
Auto Response Settings
Never Send
Send One Time Only
Send Every Time
Transmitted Data History
Data Logging is OFF (F12 to start)

Data to send (Hex and Decimal values must be separated by a space between each byte)

Treat Input Data As: Hexadecimal

Auto Response Settings: Never Send

Received Data History in ASCII

```
F▌▌ßîïCE▌▌ßñäCF▌▌ßäïCE▌▌ßäï E▌▌ßÛïCE▌▌ßäá
```

Received Data History in Hexadecimal

```
46 1F 07 DF EE EF 80 00 02 00 00
45 1F 07 DF F1 E1 80 00 01 00 00
46 1F 07 DF E2 EF 80 00 02 00 00
45 1F 07 DF E2 EF A0 00 00 00 00
45 1F 07 DF DA EF 80 00 01 00 00
45 1F 07 DF E2 E1 A0 00 00 00 00
```

Log File: (None)

Figure 4-14: Status register values of Philips TDA8044A

Besides the technical approach to acquire evidence that proves the conformity of the DVB-S channel encoder, a TV set can also be used. As this verification process uses 'live' digital TV multiplex, TV programmes can be decoded and displayed on the TV screen if the encoding standard is conformed to. Therefore, with the TV screen showing decoded digital TV programmes during the process, the designed channel encoder is validated.

4.9 MWS Campus Network Trial Concept Test

The MWS campus network trial is a 42 GHz test platform deployed in the university to explore new applications for 42 GHz MWS. At present, the existing interactive service uses cable technology that is readily available as return channel. A prototyped broadband access network is being developed at NTU to replace it with a new system configuration that is based on encapsulation of Ethernet network data in MPEG-2 TS with the introduction of an Ethernet adapter [64]. A trial was performed based on the conceptual configuration using the DVB-ENC.

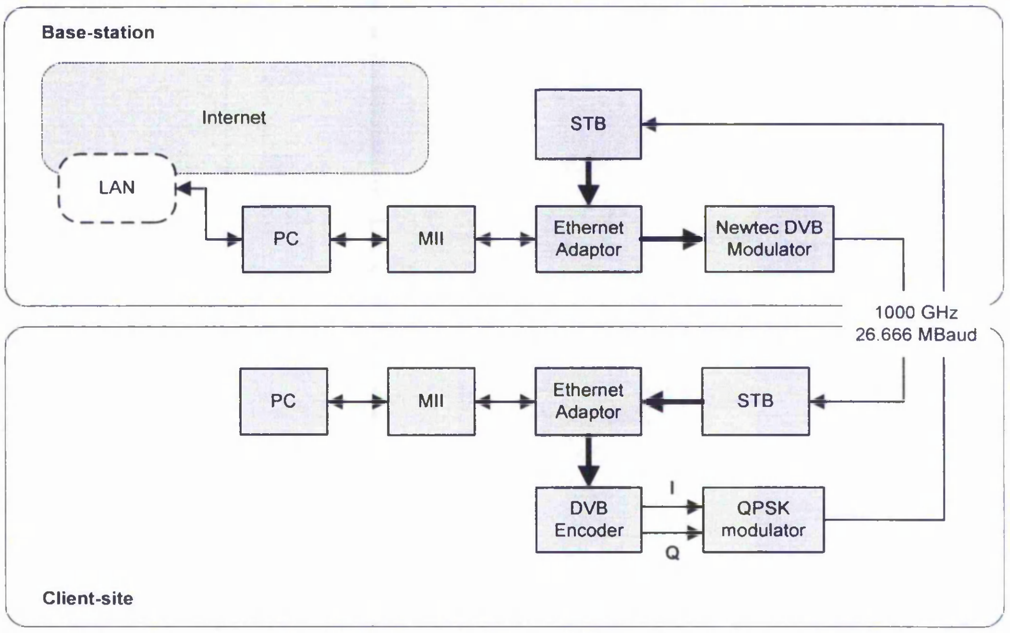
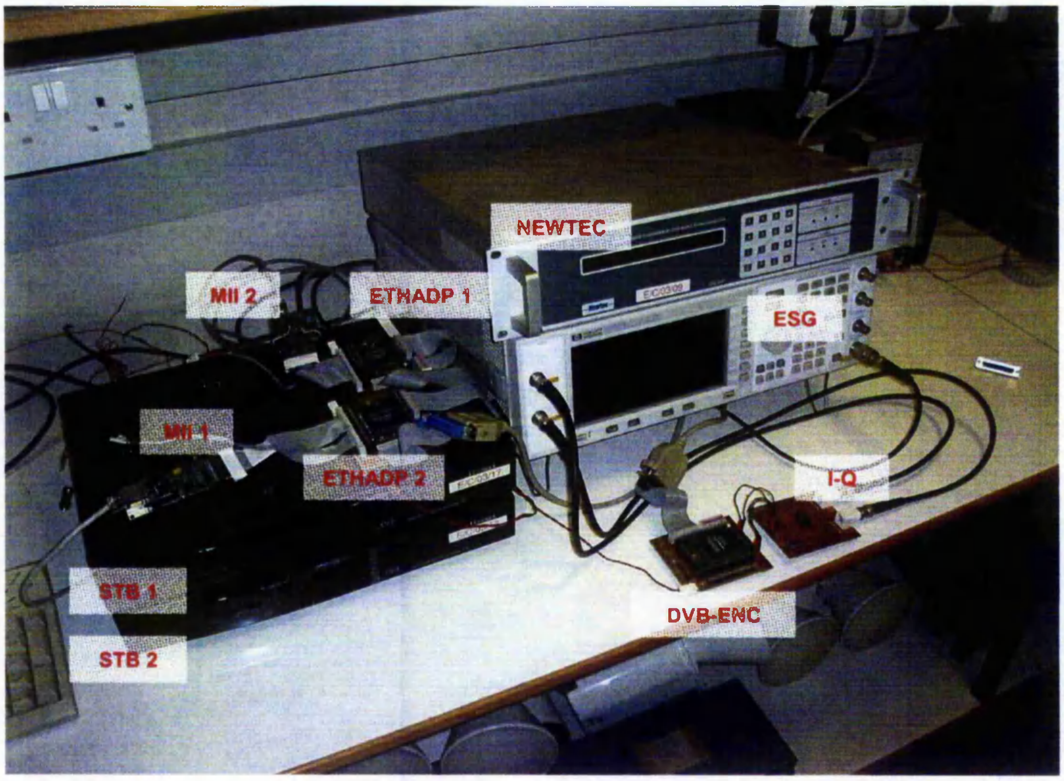


Figure 4-15: Configuration of prototyped system test

As illustrated in Figure 4-15, the set-up can be separated into base-station and client-site. At the base-station, Ethernet network data from the university network was bridged into the trial network using a PC with two network interface cards (NICs). A Media Independent Interface (MII) module was used to connect the network data to the Ethernet adapter. The Newtec DVB-S modulator was used to encode and modulate the MPEG-2 transport multiplex output from the Ethernet adapter. At the client-site, a STB was used to demodulate and recover the MPEG-2 transport multiplex. At the DVB-SPI of the STB, the MPEG-2 TS was tapped to the Ethernet adapter module. Through a MII module, Ethernet network data is connected to NIC of a client PC using Category-5 (CAT-5) cable. Aiming for an inexpensive system, the DVB-ENC was used to encode the MPEG-2 transport multiplex output of the return channel from the Ethernet adapter, the I and Q signals were QPSK modulated with a 1000 MHz carrier.

With DVB-ENC successfully encoding at 26.666 MBaud with an FEC code rate of 3/4, this test has demonstrated one of the aims of the research by replacing the bulky and expensive Newtec DVB-S modulator. At such symbol rate, the DVB-ENC is capable of supporting network data rate of up to 36 Mbps. It can be set up as part of either the transmitting-path or returning-path of the system to provide channel coding prior to transmission. Being cost effective and having a small footprint, the DVB-ENC can potentially be deployed as part of the returning-path of a client system. The system can then be easily deployed at more trial sites within the coverage cell. The success of this experiment has therefore demonstrated a novel MPEG-2 based data communications system designed, developed and implemented on an FPGA using VHDL.



STB 1 & 2	= Humax F1-FOX Set-Top-Box
MII 1 & 2	= Media Independent Interface
ETHADP 1 & 2	= Ethernet Adapter module
NEWTEC	= Newtec NTC2063 professional DVB modulator
ESG	= Agilent Technologies ESG family signal generator
DVB-ENC & I-Q	= Designed DVB-S channel encoder with I and Q output interface module

Picture 4-2: Set up of prototyped system test

5 Conclusions and Further Work

This research thesis concludes with summary of the work done. Issues worthy of further study and work are also indicated.

5.1 Summary and Conclusion

This thesis has documented a research about FPGA design and development of a DVB based channel encoder using VHDL for 42 GHz MWS. This research work has focused on investigations into a hardware system that can be used as a generic platform for applying coding schemes to the MWS campus network trial system set up in the university.

This research has achieved its main aims. The research started by investigating on various hardware technology that is available in-house and in the market as well as identifying a suitable development platform to prototype a flexible hardware that can be used to apply a range of channel coding schemes for the NTU Campus Network Trial. As a result, FPGA development was singled out as the best approach due to legacy support and facilities that are already available in the university and its suitability for rapid prototyping. A selection of commercial IP cores was also examined. They were simulated with stimuli to identify requirements and suitability for the channel encoder. The Reed-Solomon encoder IP core was implemented as part of the system with a 'zero-adding' algorithm that was designed to re-condition the MPEG-2 data stream. All modules required by the DVB-S standard channel encoding except the Reed-Solomon encoder were successfully design and developed using Altera Quartus II EDA software. They include the transport multiplex adaptor, randomizer and punctured convolutional encoder. Additional modules were designed to improve configurability of the channel encoder to allow each module to be bypass-able. Each module was simulated to verify its operations. The modules were then integrated, synthesised and programmed onto an Altera Cyclone family FPGA. The completed hardware was connected as part of the 42 GHz MWS campus network trial and successfully tested to operate as part of the end-to-end system.

Investigations were made into FPGA hardware technology that offers low-cost hardware development, as no NRE costs are involved. The reconfigurable capabilities of an FPGA made it an ideal development platform as it allows system designs to be programmed into the hardware. Changes to the designs can be reprogrammed easily. The development of the channel encoder is based on DVB-S standard published by ETSI. The standard was identified for its robustness and reliability. Using a top-down design approach, the system was broken into separate blocks with distinctive functionalities. The IP core development route was decided for the Reed-Solomon encoder to reduce the development cycle. The rest of the functional blocks were successfully designed and implemented using VHDL. Each functional block in the system was coded to be bypass able itself, including the RS encoder IP core. This allows each coding process to be selectively switched on or off in order to alter the coding scheme.

Each functional block was verified using HDL simulation on the Altera Quartus II software. On occasions, Simulink modelled simulation outputs were correlated with traces of the simulated modules. The complete system was integrated and compiled, targeting the Altera Cyclone EP1C6Q240C6 FPGA device. A total of 1,461 LEs and 15,616 memory bits were used, utilising 24% and 40% of the resources respectively. One of the two PLLs was also applied.

The implemented channel encoder was connected as part of the complete end-to-end campus network trial broadcasting configuration to perform various verification tests. An emulated MPEG transport stream and 'live' digital TV multiplex extracted from digital TV transmission in real-time were successfully re-transmitted over the modelled MWS. The status register output from the Philips TDA8044A decoder chip in the satellite STB was used as evidence to confirm that all stages of FEC were locked.

The prototyped MWS campus network trial system was set up to incorporate the implemented channel encoder as part of the concept test. The return channel of the MPEG based interactive service was encoded by the channel encoder interfacing with an Ethernet adapter module. The test has successfully demonstrated its operation, encoding at the rate of 26.666 MBaud with an FEC code rate of 3/4, supporting a network data rate of up to 36 Mbps.

Besides operating at an FEC code rate of 3/4, the system can also be configured to encode at 1/2 rate with the symbol rate of 40.000 MBaud using the same PLL output clock frequencies. As the receiver STB is only capable of decoding at a maximum symbol rate of 32.767 MBaud [77], a system wide test was not performed. Based on the current implementation that uses one PLL to generate the interface and system clocks, the ideal 1.085 (204/188) times difference between the clock frequencies cannot be achieved. This is due to requirements for the Cyclone PLL to generate the output frequencies that are not met as the VCO of the Cyclone PLL would be operating outside of its capacities.

The success in the development of the FPGA hardware has cemented the realisation of an inexpensive DVB based channel encoder. Existing expensive professional frequency translation equipment deployed at trial sites can be replaced and the number of sites can be increased. The system can also provide a good test platform for establishing an optimised coding scheme for an improved terrestrial broadband fixed wireless access. The level of flexibility allows customisation of all the functional blocks, where they can be independently disabled. The FPGA system has also proven to be more compact compared to the existing channel coding devices, which ensures easy deployment of the system to new client sites.

5.2 Further Research Work

The 3/4 rate puncture code was implemented to improve efficiency of the coding scheme, instead of simply convolutional encoded. The puncturing module was designed to allow its puncturing rate to be changed from 1/2 to 3/4 as proof of concept towards developing a reconfigurable channel coder for 42 GHz MWS. The 3/4 rate puncture code was implemented first due to its simplicity and as it was the setting used for the existing campus network trial. As presented in Table 3-2, puncturing patterns are defined by DVB for the code rates 1/2, 2/3, 3/4, 5/6, and 7/8 [56]. An efficient puncturing algorithm can potentially be designed to allow variable code rates, with minimum complexity and space used on an FPGA. Therefore, further design work is suggested for the implementation of the rest of the puncturing scheme to enable maximum rate efficiency and flexibility of the channel coder.

In view of the effects of changing weather patterns on 42 GHz MWS links, further work can be done on developing algorithms to adapt to changes in attenuation levels by changes in encoding schemes, such as puncturing rate. The Phillips TDA8044A chip [76] that is used by the Humax F1-FOX satellite STB [77] provides a rough estimation of bit error rate (BER) to the STB and can be read via the I^{2}C-bus. Previous research work has successfully acquired these values for BER measurements [59] and hence can be further explored to potentially develop an adaptive system that changes the FEC by changing the puncturing rate to achieve a minimum BER at any given time.

With the BER estimations, the level of FEC can be evaluated. Using the return link, requests of change of FEC rate can be sent to the base-station. Information on the new FEC setting can be integrated into the header of data packets at the transmitting-end. At the receiving-end, modifications can potentially be done to enable detection of the embedded information to match the requested FEC setting.

Development of this system in VHDL enables the DVB-S channel encoder system to be integrated into devices, with systems such as the Ethernet adapter, as long as its timing, I/O ports and space requirements are satisfied. System integration with a medium access control (MAC) layer is a provision towards realising an adaptive encoder for 42 GHz MWS. Besides, using a single device would eliminate the use of some interfacing and reduce the size of the system.

Further work on hardware could potentially tap the full capabilities of the Cyclone family FPGA. Instead of using the Cyclone development board, a fully customised PCB could be developed for the FPGA device to increase its resources, such as additional I/O pins, and global clock inputs. An additional global clock input could allow further exploitation of the Cyclone PLL. The interface clock can be synchronised with interfacing external systems. This potentially allows the interface clock to be changed with the adaptive FEC rate and also allows the channel encoder to interface with other systems more effectively. One of such systems is the Ethernet adapter. Integration of such system could pave the way towards an inexpensive wireless access network SoC.

One of the advantages of targeting an FPGA device is its ability to be re-configured on-the-fly. This can be done by having the entire designed system and all of its possible options programmed onto the FPGA with the controls to change the inactive options to active dynamically. This technique is referred to as Multiple-context Configuration Memory [99]. This technique is fast to switch between the options, but with all possible options implemented, it requires a large silicon area. Besides that, another way for dynamic re-configuration is to have the FPGA partitioned, separating the standard system and the variable section. This is made possible with the Partial Configuration technique [99]. With the different options stored in memory, the variable section of the FPGA can be re-configured with the selected option. Although this technique is more space optimised, the speed of re-configuration of the variable section is directly proportional to the size to be re-configured. In-depth investigation into these dynamic re-configuration techniques and FPGA partitioning is suggested to fully exploit the capabilities of an FPGA.

References

[1] Nordbotten, "LMDS Systems and their Application," *IEEE Communications Magazine*, pp. 150-154, Jun. 2000.

[2] Smith, *LMDS*, McGraw-Hill Professional Telecom Publication, Aug. 2000.

[3] *Communications Liberalisation in the UK*. London: Department of Trade and Industry, 2001.

[4] eEurope 2005: An information society for all. Serville: European Council, 2002.

[5] *UK online: the broadband future*. London: Cabinet Office, 2001.

[6] Broadband Stakeholder Group, *Broadband in Rural Areas - Broadband Stakeholder Group Submission to EFRA Committee*, Apr. 2003, http://www.broadbanduk.org/ [Accessed 25 Aug. 2004]

[7] A.J. Paulraj, et al., "Fixed Broadband Wireless Access: State of the Art, Challenges, and Future Directions," *IEEE Communications Magazine*, pp. 100-108, Jan. 2001

[8] V.I. Roman, "Frequency Reuse and System Deployment in Local Multipoint Distribution Service," IEEE Personal Communications, pp. 20-27, Dec. 1999.

[9] European Radiocommunications Committee, (ERC/DEC/(99)/16), ERC Decision (99)16 of 1 June 1999 on the withdrawal of the ERC Decision (96)05 "Decision on the harmonised frequency band to be designated for the introduction of the Multipoint Video Distribution Systems (MVDS)", Jun. 1999.

[10] European Radiocommunications Committee, (ERC/DEC/(99)/15), ERC Decision (99)15 of 1 June 1999 on designation of the harmonised frequency band of the 40.5 to 43.5 GHz for the introduction of Multimedia Wireless Systems (MWS) including Multipoint Video Distribution Systems (MVDS)", Jun. 1999.

[11] V.I. Mostovoy, "Cellular Television – A High Technology of Broadband Wireless Access," *in 12th International Crimean Confrernce for Microwave & Telecommunications Technology (CriMiCo'2002)*, pp. 3-8, Sep 2002.

[12] P. Mähönen, T. Saarinen and Z. Shelby, "Wireless Internet over LMDS: Architecture and Experimental Implementation," IEEE Communications Magazine, pp. 126-132, May 2001.

[13] European Telecommunications Standards Institute, ETSI EN 301 997-1, Transmission and Multiplexing (TM); Multipoint equipment; Radio equipment for

use in Multimedia Wireless Systems (MWS) in frequency hand 40,5 GHz to 43,5 GHz; Part 1: General requirements, Sept. 2003

[14] [S.A. Kravchuk and M.E. Ilchenko, "Broadband Wireless Access Systems; Terms and Definitions," *in 12th International Crimean Confrernce for Microwave & Telecommunications Technology (CriMiCo'2002)*, pp. 55, Sep 2002.

[15] H. Linder, H.D. Clausen and B. Collini-Nocker, "Satellite Internet Services Using DVB/MPEG-2 and Multicast Web Caching," *IEEE Communications Magazine*, pp. 156-161, Jun. 2000.

[16] R. Germon, et.al., "42 GHz Multimedia Wireless System Campus Trial," in *IEE Seminar for New Access Network Technologies*, pp. 6/1-6/5, 2000.

[17] IEEE Computer Society, IEEE Std 802.16.2-2001, IEEE Recommended Practise for Local and Metropolitan Area Networks: Coexistence of Fixed Broadband Wireless Access Systems, 2001.

[18] G. Fairhurst, S.L. Pang and P.S. Wan, "Smart Codec: An Adaptive Packet Data Link," *in IEE Proceedings: Communications*, vol. 145, no. 3, pp. 180-185, Jun 1998.

[19] W. Webb, "Broadband Fixed Wireless Access as a Key Component of the Future Integrated Communications Environment," *IEEE Communications Magazine*, vol. 39, no. 9, pp. 115-121, Sep 2002.

[20] Altera Corporation, "Broadband Fixed Wireless Applications in Cyclone Devices," *http://www.altera.com/products/devices/cyclone/features/cyc-fixed_wireless.html* [Accessed 7 Sep. 2004]

[21] R.H. Day, R. Germon and B.C. O'Neill, "A Real Time Digital Signal Processing Solution for Radar Pulse Compression," *in Proceedings of IEE Colloquium on Digital Filters: An Enabling Technology*, vol. 252, pp. 6/1-6/5, 1998.

[22] M. Cummings and S. Haruyama, "FPGA in the Software Radio," *IEEE Communications Magazine*, vol. 37, no. 2, pp. 108-112, Feb 1999.

[23] Wikipedia, "Electronic Design Automation", *http://en.wikipedia.org/wiki/Electronic_design_automation* [Accessed on 15 Nov 2004].

[24] Jansen, et al., *The Electronic Design Automation Handbook*, Boston: Kluwer Academic Publisher, 2003.

[25] Electronic Industries Alliance, "EDIF Introduction," *http://www.edif.org/introduction.html* [Accessed on 15 Nov 2005].

[26] M.N. Cirstea, "Modern Electronic Design Automation Techniques (based on Hardware Description Languages) Applied to Power Electronic Systems Holistic Modelling," *in Tutorials Volume of IEEE International Symposium on Industrial Electronics (ISIE'2003)*, pp. 23-52, Jun 2003.

[27] Altera Corporation, "Quartus II Software – Design to Win," *http://www.altera.com/products/software/products/quartus2/* [Accessed on 15 Nov 2004].

[28] Altera data sheet, *MAX+PLUS II Programmable Logic Development System & Software Data Sheet*, A-DS-MPLUS2-08, ver. 8.0, Altera Corporation, Jan1998, http://www.altera.com/literature/ds/dsmii.pdf [Accessed on 16 Nov 2004].

[29] Altera Corporation, "Quartus II Software & Quartus II Web Edition Software Feature Comparison," *http://www.altera.com/products/software/products/quartus2web/features/sof-quarweb_features.html*, [Accessed on 17 Nov 2004].

[30] Altera user guide, *Introduction to Quartus II Manual*, version 4.1, Altera Corporation, Jun 2004, http://www.altera.com/literature/manual/intro_to_quartus2.pdf, [Accessed on 17 Nov 2004].

[31] Altera literature, *Intellectual Property Selector Guide*, SG-IP-3.0, ver. 3.0, Altera Corporation, Mar 2003, http://www.altera.com/literature/sg/sg_ip.pdf [Accessed 26 Sep 2004].

[32] Mentor Graphics Corporation, "ModelSim," *http://www.modelsim.com/* [Accessed on 20 Nov 2004].

[33] D.E. Ott and T.J. Wilderotter, *A Designer's Guide To VHDL Synthesis*, Massachusetts: Kluwer Academic Publishers, 1994.

[34] R. Day, *A DSP Controller for a Low Cost Radar Interface*, Ph.D Thesis, Nottingham Trent University, 1999.

[35] J.H. Ng, *Message Routing Interface for Multiprocessor Networks*, Ph.D Thesis, Nottingham Trent University, 2001.

[36] M. Whitbread, "IP for FPGA – Where, What and How Much?" *Embedded System Engineering*, pp. 36-42, vol. 12.6, Sep 2004.

[37] Edwards, "Bug to Basics," *IEE Review*, pp. 38-41, vol. 50, no. 3, Mar 2004.

[38] P. Lindermann and B. Finch, "IP Cores for FPGAs," Embedded System Engineering, pp. 49-50, vol. 12.6, Sep 2004.

[39] Altera Corporation, "Designing with Altera Intellectual Property Megafunctions," *http://www.altera.com/products/ip/design/ipm-design.html* [Accessed on 20 Nov 2004].

[40] Altera Corporation, "AMPP IP Providers," *http://www.altera.com/products/ip/ampp/ampp1.html* [Accessed on 20 Nov 2004].

[41] Altera application note, *OpenCore Plus Evaluation of Megafunctions*, AN-320, ver. 1.2, Altera Corporation, June 2004, http://www.altera.com/literature/an/an320.pdf [Accessed on 20 Nov 2004].

[42] Email from R. Marcoccia, Altera University Program, Altera Corporation, RMARCOCC@altera.com, 10 May 2005.

[43] Z. Salcic and A. Smailagic, Digital Systems Design and Prototyping using Field Programmable Logic. USA: Kluwer Academic Publishers, 1997

[44] K. Morris, "All is Not SRAM - A survey of flash, antifuse, and EE programmable logic," *FPGA and Programmable Logic Journal*, Feb 2004, http://www.fpgajournal.com/articles/sram.htm [Accessed on 10 Nov 2005].

[45] M.J.S. Smith, *Application-Specific Integrated Circuits*. USA: Addision-Wesley, 1997.

[46] R. Ball, "Shifting architectures: CPLDs versus FPGAs," *Electronics Weekly*, pp. 20, no. 2162, 15 Sep 2004.

[47] S. Bremec, R. Uršič and U. Mavrič, "Advantages of Implementing Digital Receivers in Field Programmable Gate Arrays (FPGA)," *6th European Workshop on Beam Diagnostics and Instrumentation for Particle Accelerators*, 2003, http://www.i-tech.si/support-conference.html [Accessed on 10 Nov 2004].

[48] Altera Corporation, "Cyclone: The Lowest-Cost FPGA Ever," *http://www.altera.com/products/devices/cyclone/cyc-index.jsp* [Accessed on 7 Sep. 2004]

[49] Newtec application note, *NTC/2080/APN03 Application Note: Choice of QPSK - 8PSK or 16QAM in DVB Satellite Links*, Newtec, Jul 1999, http://www.newtec.be/appnotes/ntc.2080.apn03.pdf [Accessed on 8 Sep 2004]

[50] Altera Corporation, "Altera University Program," *http://www.altera.com/education/univ/unv-index.html*, [Accessed in 8 Sep 2004]

[51] Altera data sheet, *FLEX 10KE PCI Development Board*, A-DS-PCI-1.2 Altera Corporation, Nov 2001.

[52] Altera press release, "Altera's Cyclone FPGAs Take the Industry by Storm," Altera Corporation, Sep 2002,

http://www.altera.com/corporate/news_room/releases/releases_archive/2002/products/nr-cyclone.html, [Accessed on 15 Nov 2004]

[53] Altera white paper, *Cyclone vs. Spartan-3 Performance Analysis*, Altera Corporation, http://www.altera.com/literature/wp/wpcycsptn3pa.pdf [Accessed on 26 Sep 2004]

[54] Altera literature, *Using PLLs in Cyclone devices*, C51006-1.2, ver. 1.2, Altera Corpration, Oct. 2003, http://www.altera.com/literature/hb/cyc/cyc_c51006.pdf [Accessed on 26 Sep 2004]

[55] M. Schoeberl, "Cyclone FPGA Board," JOP Design, http://www.jopdesign.com/cyclone/index.jsp [Accessed on 10 Nov 2004].

[56] European Telecommunications Standards Institute, ETSI, EN 300 421, Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for 11/12 GHz satellite services, Aug. 1997.

[57] Grondalen, AC215/TEL/RD/R/P/D4P4/b1, *Interactive Broadband Technology Trials*, ACTS Project 215 deliverable report D4P4, Jan. 1999.

[58] International Organisation for Standardization, ISO/IEC DIS 13818-1, *Generic Coding of Moving Picture and Associated Audio Information Systems*. Jun. 1994.

[59] E.B. Lam, *42 GHz Multimedia Wireless System – Measurement and Analysis*, Ph.D. thesis, Nottingham Trent University, 2001.

[60] European Telecommunications Standards Institute, ETSI, EN 300 800, Digital Video Broadcasting (DVB); Interaction channel for Cable TV distribution systems (CATV), Jul. 1998.

[61] European Telecommunications Standards Institute, ETSI EN 301 199, Digital Video Broadcasting (DVB); Interaction channel for Local Multi-point Distribution Systems (LMDS), Jun. 1999.

[62] COCOM A/S, INA HFC head-end CC1014 – Operator and service manual, 1999.

[63] W.Y. Mun and R. Germon, "Data interface for 42 GHz DVB Multimedia Wirelss Systems (MWS)," *PREP 2003 Postgraduate Research Conference*, 2003.

[64] W.Y. Mun and R. Germon, "An Efficient Ethernet over DVB Encapsulation Technique for MWS," in Proceedings of the 5th Annual PostGraduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting, PG Net 2004, pp. 285-290, Jun 2004.

[65] I.A. Glover and P.M. Grant, *Digital Communications, 2nd Edition*. UK: Pearson Education Limited, 2004.

[66] G. Held, Data Communications Networking Devices: Operation, Utilization and LAN and WAN Internetworking, 4th Edition, UK: John Wiley & Sons Limited, 1999.

[67] Digital Video Broadcasting (DVB) Project, http://www.dvb.org.

[68] G. Xylomenos and G.C. Polyzos. "TCP and UDP performance over a wireless LAN," in Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM '99, vol. 2, pp. 439-446, Mar. 1999.

[69] Amir, et.al., "Efficient TCP over networks with wireless links," *in Proceedings of the 5th Workshop on Hot Topics in Operating Systems, HotOS-V*, pp. 35-40, May 1995.

[70] European Committee for Electrotechnical Standardization, CENELEC, EN 50083-9, Cable networks for television signals, sound signals and interactive services; Part 9: Interfaces for CATV/SMATV headends and similar professional equipment for DVB/MPEG-2 transport streams, Jun 1998.

[71] S. Jayasimha, "Pragmatic TCM Using 8-PSK in Satellite Communications," *TechOnLine*, Apr 2002, http://www.techonline.com/community/ed_resource/feature_article/20514 [Accessed on 20 Nov 2004]

[72] M Cominettti and A. Morello, "Digital Video Broadcasting over Satellite (DVB-S): A system for broadcasting and contribution applications," *International Journal of Satellite Communications*, vol, 18, pp. 393-410, 2000.

[73] P. Fen, "Digital Television Terrestrial Broadcasting Primer, " *Online Symposium for Electronics Engineers*, Oct. 2001, TechOnLine: http://www.techonline.com/community/ed_resource/feature_article/14706 [Accessed on 10 Sep. 2004]

[74] C. Fleming, "A Tutorial on Convolutional Coding with Viterbi Decoding," *Spectrm Applications*, Jan 2003, http://home.netcom.com/~chip.f/viterbi/tutorial.html [Accessed on 20 No 2004].

[75] T. Kratochvil, "Digital Video Broadcasting channel encoding and decoding simulation," *in 4th EURASIP Conference on Video/Image Processing and Multimedia Communications*, pp. 851-856, Jul. 2003

[76] Philips Semiconductors product specification, *STB5660 (set-top-box) STB concept*, OM5721, Philips, May 1999.

[77] Humax user manual, *Humax F1-FOX Operating Manual*, HUMAX Electronics Co. Ltd., http://www.humaxdigital.com/HCSA/usersmanual.asp [Accessed 15 Oct 2004].

[78] Fujitsu data sheet, *QPSK Silicon Tuner*, MB86A15, ed. 4.1, Fujitsu Limited, 2002

[79] Philips literature, The I^{2}C-bus specification version 2.1, Philips Semiconductor, Jan 2000, http://www.semiconductors.philips.com/buses/i2c/ [Accessed 15 Oct 2004].

[80] Newtec user manual, *Determining the Maximum Datarates in (De)Modulaters*, NTC/2063/APN04, Newtec, Jul 1998, http://www.newtec.be/appnotes/ntc.2063.apn04.pdf [Accessed 21 Sep. 2004]

[81] Altera Megafunctions user guide, *Single & Dual-Clock FIFO Megafunctions User Guide*, UG-MFNALT_FIFO--2.0, Altera Corporation, Sep 2004, http://www.altera.com/literature/ug/ug_fifo.pdf [Accessed on 5 Oct 2004].

[82] Altera application note, *Metastability in Altera Devices*, AN-42, ver. 4, Altera Corporation, May 1999, http://www.altera.com/literature/an/an042.pdf, [Accessed on 7 Dec 2004].

[83] Altera Corporation, "Library of Parameterised Modules (LPM)," *http://www.altera.com/products/software/products/legacy/maxplus2/sfw-lpm.html* [Accesed on 7 Oct 2004].

[84] C.C. Bissell and D.A. Chapman, *Digital Signal Transmission.* UK: Cambridge University Press, 1992.

[85] S. Lin and D.J. Costello, Error Control Coding: Fundamentals and Applications. USA: Prentice-Hall, 1983.

[86] Altera Megafunctions user guide, Reed-Solomon Compiler User Guide, UG-RSCOMPILER-3.9, ver. 3.6, Altera Corporation, Jul 2004, http://www.altera.com/literature/ug/rs-compiler_ug.pdf [Accessed on 5 Oct 2004].

[87] H. Sari, "Some Design Issues in Local Multipoint Distribution Systems," *1998 URSI International Symposium on Signals, Systems, and Electronics, ISSSE 98*, pp. 13-19, Sep 1998.

[88] G.C. Clark, Jr and J.B. Cain, *Error-Correction Coding for Digital Communications.* USA: Plenum Press, 1981.

[89] S. Haykin, *Digital Communications.* USA: John Wiley & Sons, 1998.

[90] G.C. Clark, Jr., and J.B. Cain, *Error-Correction Coding for Digital Communications.* USA: Plenum Press, 1981.

[91] The Mathworks, http://www.mathworks.com [Accessed on 1 Nov 2004].

[92] R.H. Bishop, Modern Control Systems Analysis and Design using MATLAB and Simulink. USA: Addison Wesley Longman, 1997.[

[93] STMicroeletronics data sheet, *QPSK/BPSK Link*, STV0299, STMicroelectronics, Nov 1999.

[94] Fujitsu data sheet, *QPSK Silicon Tuner*, MB86A15, ed. 4.1, Fujitsu Limited, 2002

[95] Philips data sheet, *Satellite Demodulator and Decoder*, TDA8044/TDA8044A, Philips Semiconductor, Feb 1998.

[96] Agilent Technologies user's guide, *Agilent Technologies ESG Family Signal Generators*, part no. E4400-90323, Agilent Technologies, Apr 2000.

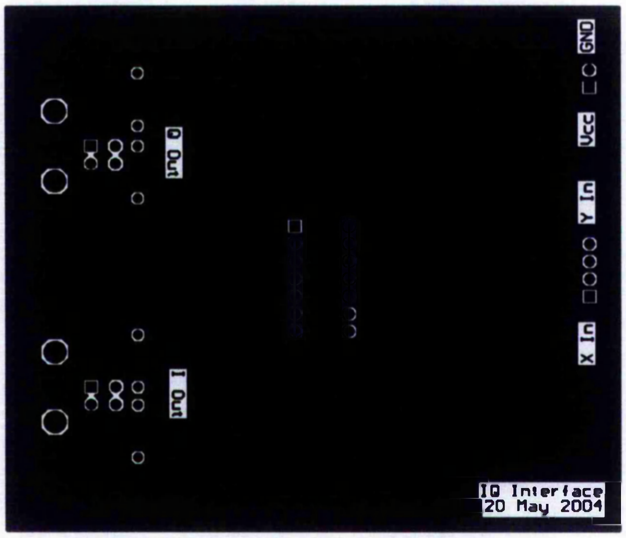
[97] Wikipedia, http://en.wikipedia.org/wiki/Gerber_File [Accessed on 20 Oct 2005].

[98] Altera application note, *High-Speed Board Designs*, A-AN-75-4.0, ver. 4.0, Altera Corporation, Nov 2001.

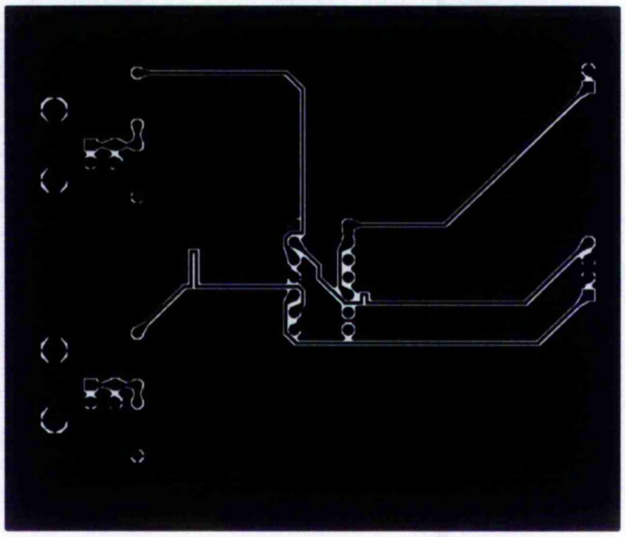
[99] Anon, *Dynamically Reconfigurable Devices & Technology*, Bournemouth University's DRHW WWW Library, Dec 2000, http://dec.bournemouth.ac.uk/drhw_lib/technology.html [Accessed on 16 Nov 2005].

Appendix I: I and Q Interface Module PCB Design

Top Layer:

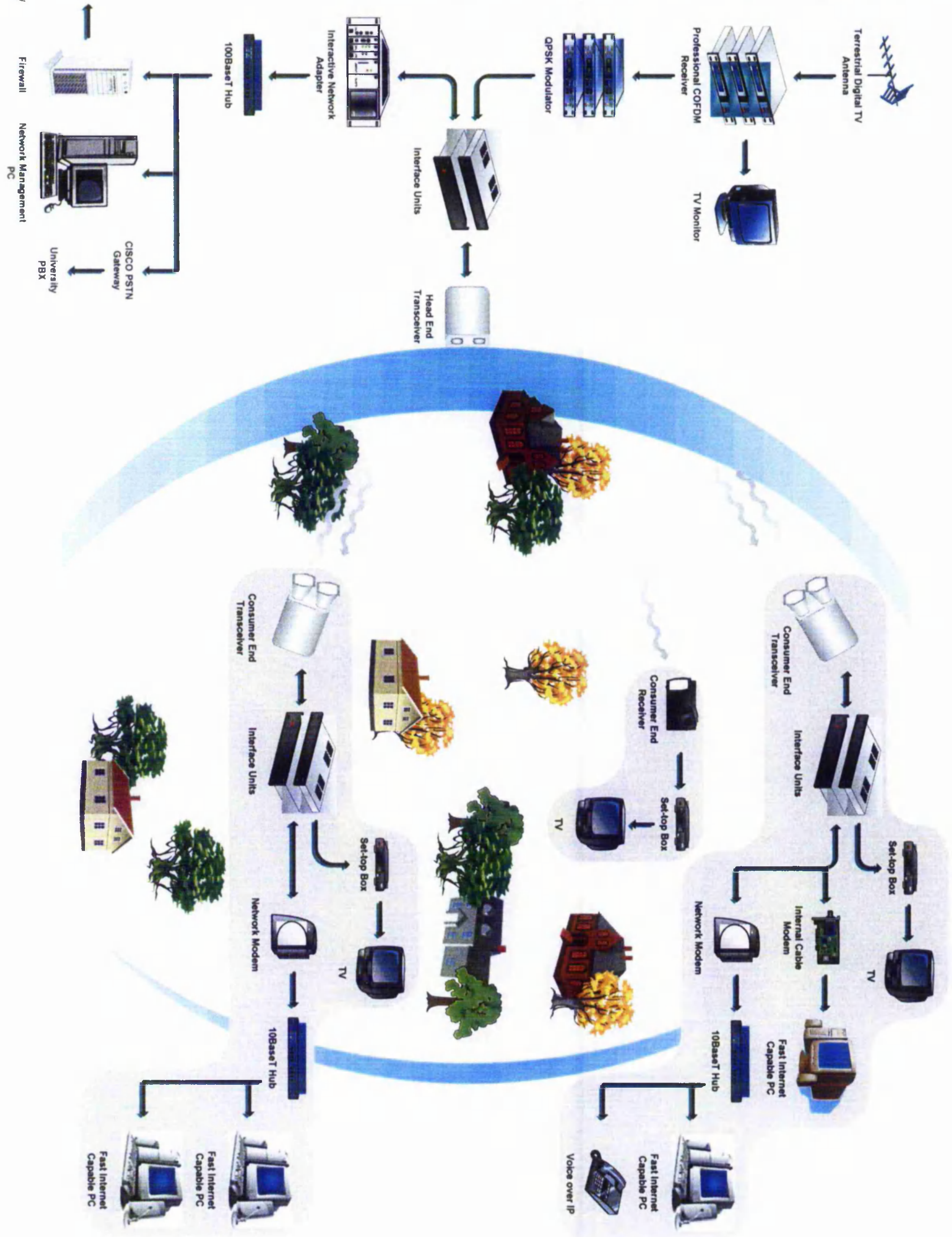


Bottom Layer:



Appendix II: NTU Campus Network Trial - Overview

BROADBAND FIXED WIRELESS ACCESS CAMPUS NETWORK

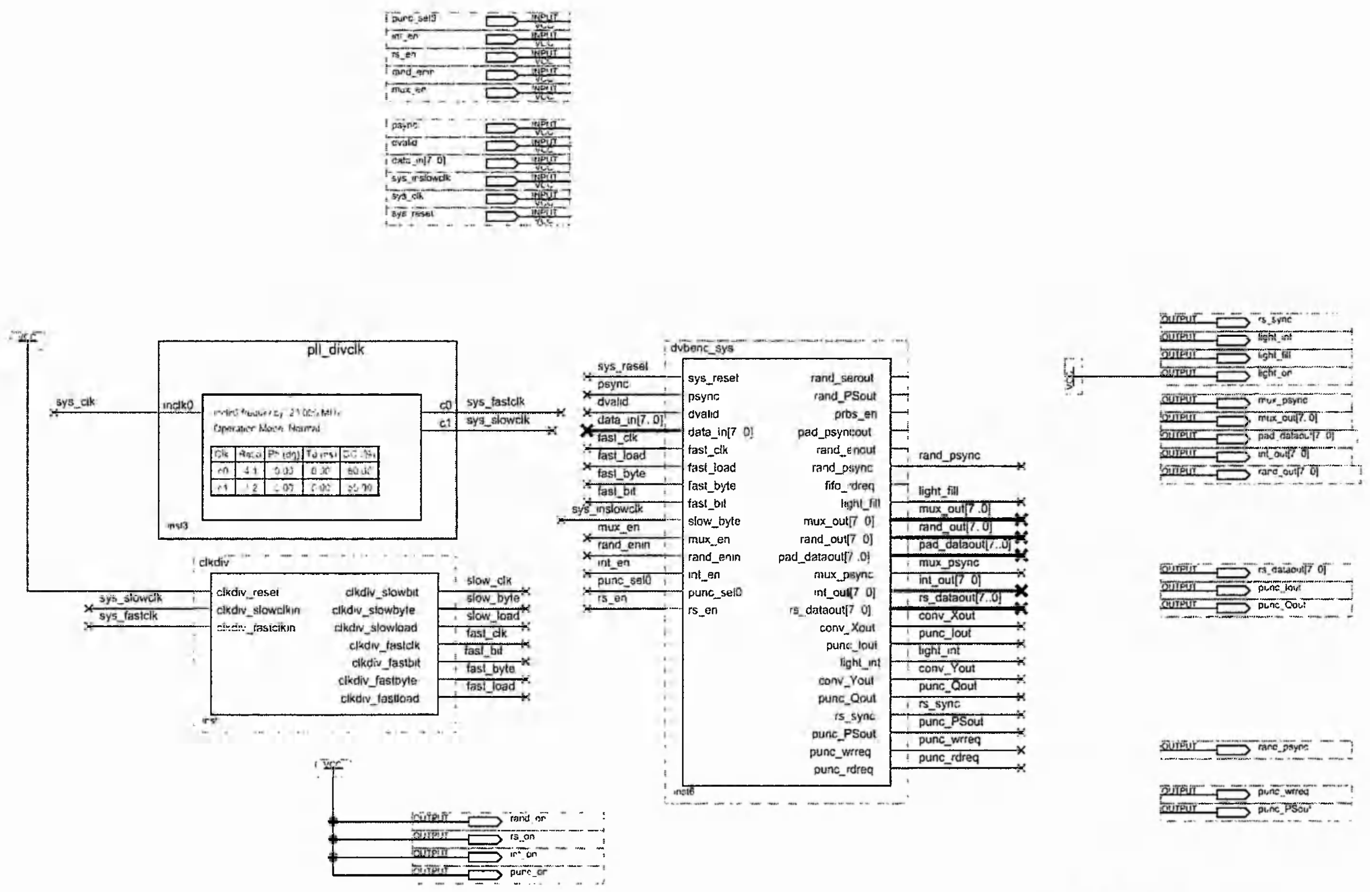


Appendix III: Block Diagram & Source Code Listing

Module: dvbenc_burn

Description: DVB Encoder top-level interface design

Date: June 30, 2004 dvbenc_burn.bdf* Project: dvbenc_burn

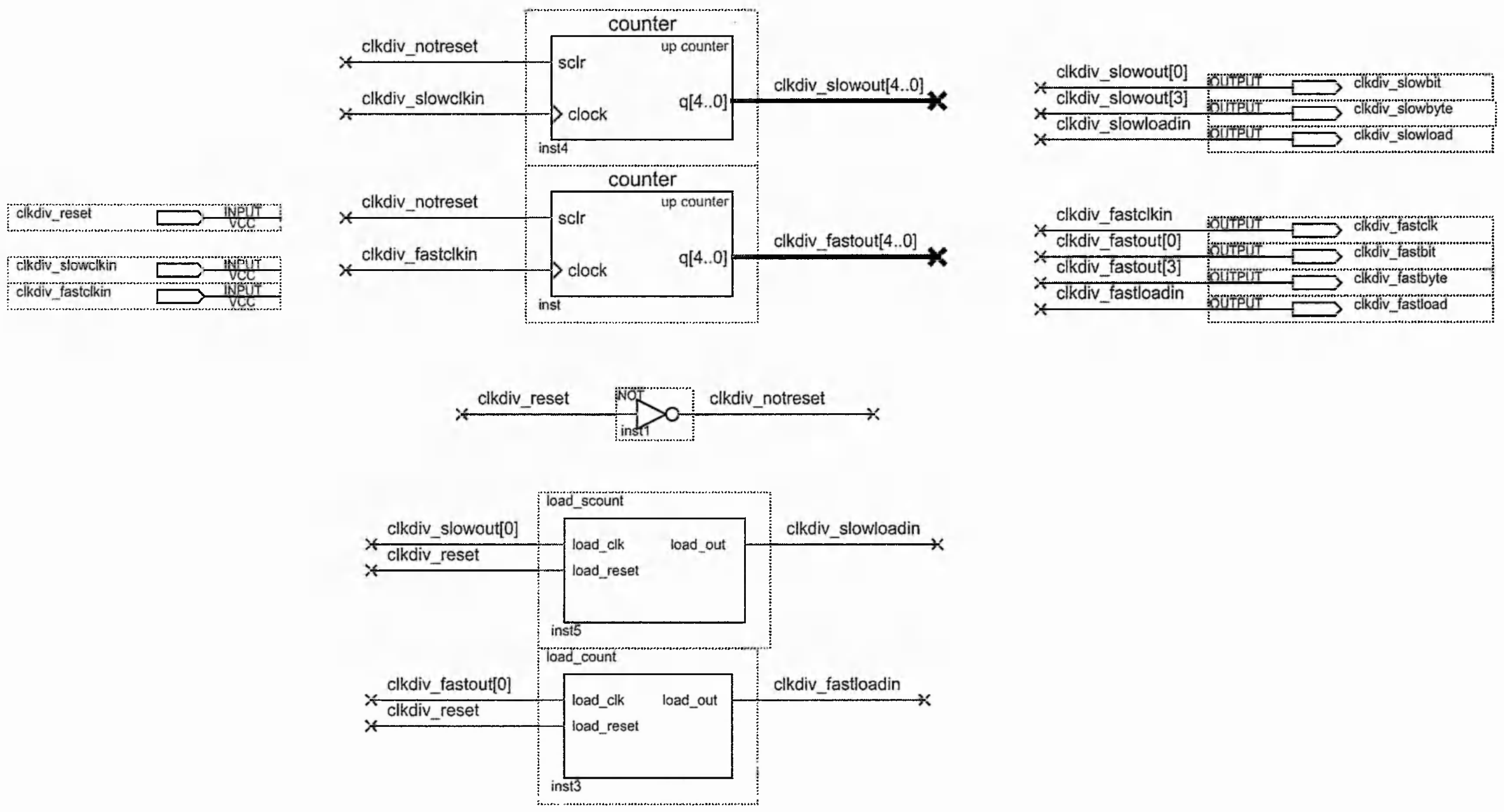


Page 1 of 1

Module: clkdiv

Description: System clock division

Date: March 23, 2006 ../clkdiv/clkdiv.bdf* Project: dvbenc_burn



Page 1 of 1 Revision: dvbenc_burn

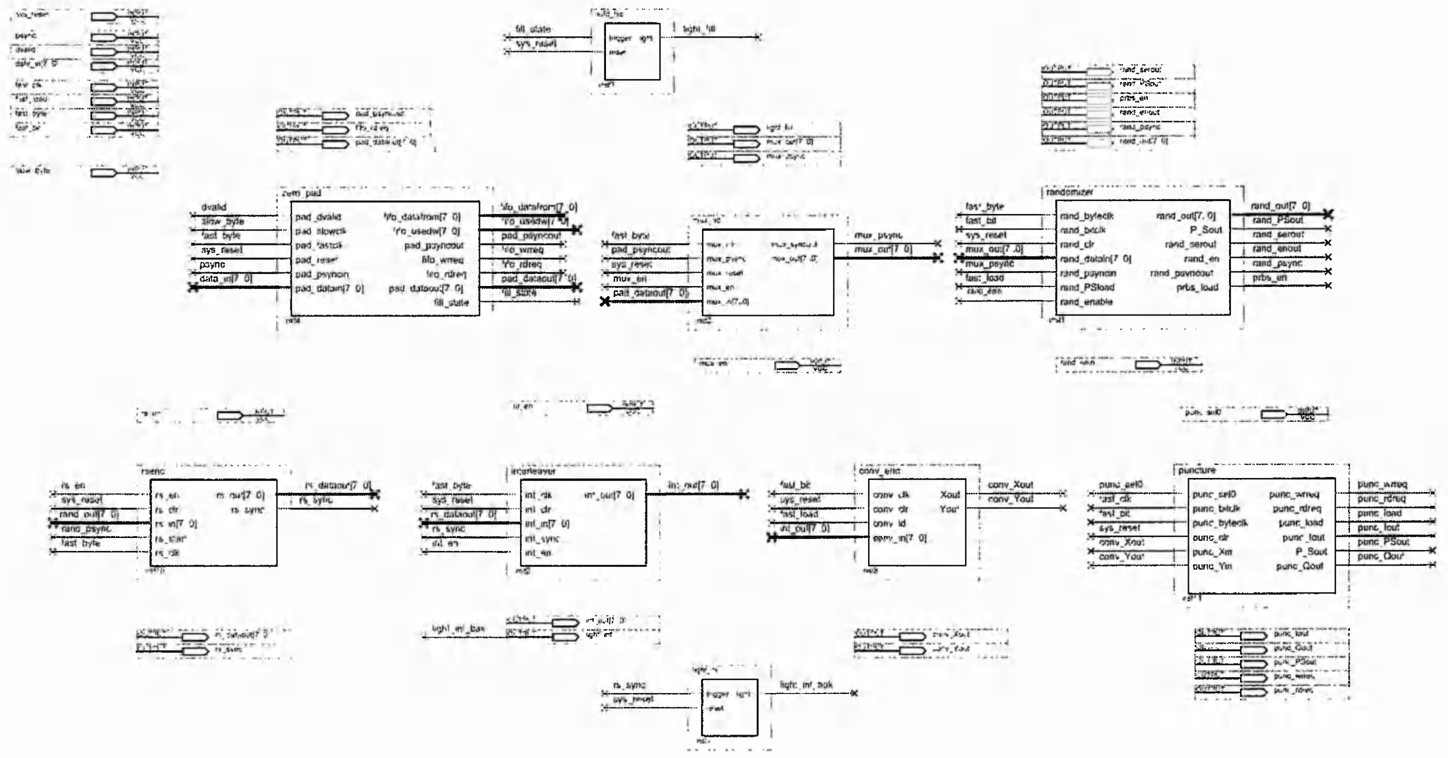
```
-- author: kaohsiung chuah
-- date: 17/06/2003
-- filename: load_count.vhd
-- description: this generates the load signal for
--              parallel to serial converter

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY load_count IS
    PORT
    (
        load_clk    : IN STD_LOGIC;
        load_reset  : IN STD_LOGIC;
        load_out    : OUT STD_LOGIC
    );
END load_count;

ARCHITECTURE rtl OF load_count IS
    SIGNAL load_sig : STD_LOGIC;
BEGIN
    PROCESS (load_clk, load_reset)
        VARIABLE clk_count  : INTEGER RANGE 0 TO 7;
    BEGIN
        IF (load_reset = '0') THEN
            clk_count := 0;
        ELSIF (load_clk'EVENT AND load_clk = '1') THEN
            clk_count := clk_count + 1;
            -- generate load signal load signal
            -- change clk_count value to control output timing
            IF (clk_count = 4) THEN
                load_sig <= '1';
            ELSE
                load_sig <= '0';
            END IF;
        END IF;

        IF (load_reset = '0') THEN
            load_out <= '0';
        ELSIF (load_clk'EVENT AND load_clk = '0') THEN
            load_out <= load_sig;
        END IF;
    END PROCESS;
END rtl;
```

```
-- author: kaohsiung chuah
-- date: 17/06/2003
-- filename: load_scount.vhd
-- description: this generates the load signal for
--              parallel to serial converter

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY load_scount IS
    PORT
    (
        load_clk    : IN STD_LOGIC;
        load_reset  : IN STD_LOGIC;
        load_out    : OUT STD_LOGIC
    );
END load_scount;

ARCHITECTURE rtl OF load_scount IS
    SIGNAL load_sig : STD_LOGIC;
BEGIN
    PROCESS (load_clk, load_reset)
        VARIABLE clk_count   : INTEGER RANGE 0 TO 7;
    BEGIN
        IF (load_reset = '0') THEN
            clk_count := 0;
        ELSIF (load_clk'EVENT AND load_clk = '1') THEN
            clk_count := clk_count + 1;
            -- generate load signal load signal
            -- change clk_count value to control output timing
            IF (clk_count = 3) THEN
                load_sig <= '1';
            ELSE
                load_sig <= '0';
            END IF;
        END IF;

        IF (load_reset = '0') THEN
            load_out <= '0';
        ELSIF (load_clk'EVENT AND load_clk = '0') THEN
            load_out <= load_sig;
        END IF;
    END PROCESS;
END rtl;
```

Module: dvbenc_sys

Description: DVB-S encoding module integration

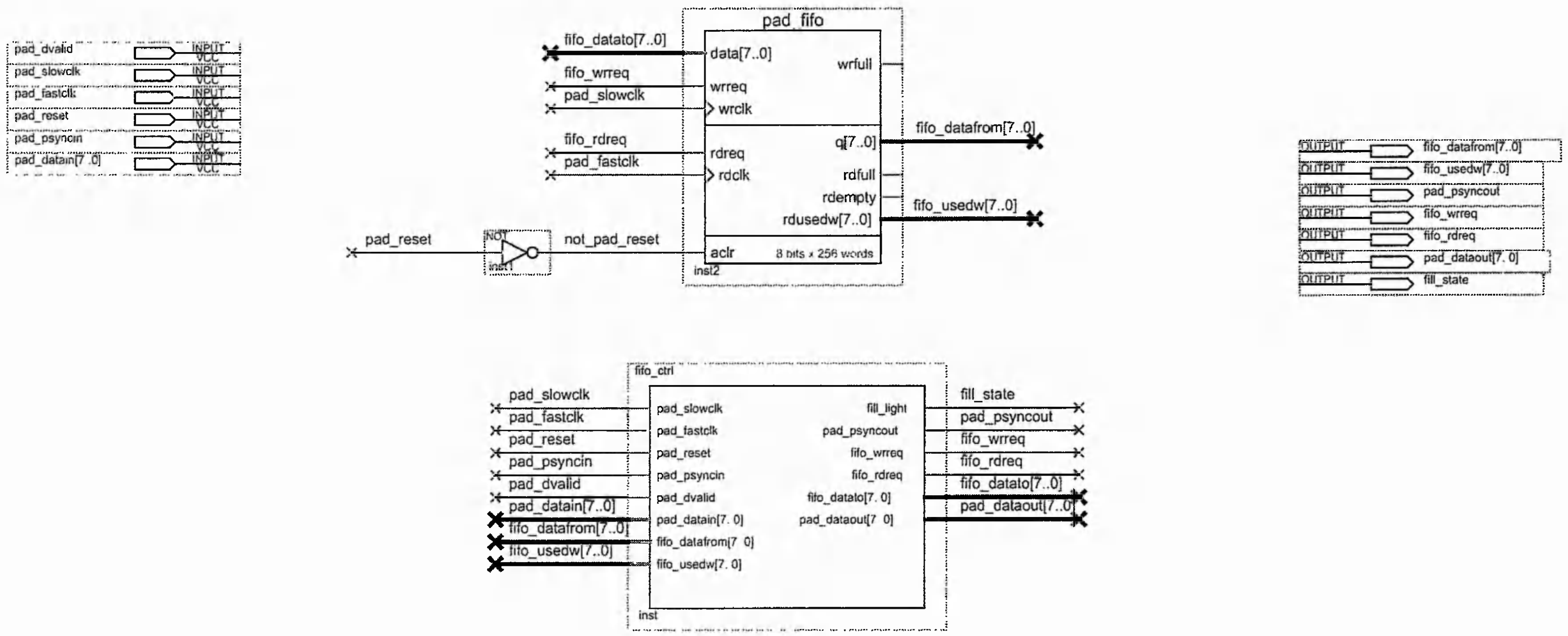
Date: March 23, 2006 dvbenc_sys.bdf* Project: dvbenc_burn



Page 1 of 1

Module: zero_pad

Description: MPEG packet reconditioning to add 16 null-bytes for Reed-Solomon encoding


pad_dvalid
pad_slowclk
pad_fastclk
pad_reset
pad_psyncin
pad_datain[7..0]
INPUT
VCC
pad_fifo
fifo_datato[7..0]
data[7..0]
wrfull
fifo_wrreq
wrreq
pad_slowclk
wrclk
fifo_rdreq
rdreq
pad_fastclk
rdclk
q[7..0]
fifo_datafrom[7..0]
rdfull
rdempty
rdusedw[7..0]
fifo_usedw[7..0]
pad_reset
NOT
inst1
not_pad_reset
aclr
8 bits x 256 words
inst2
OUTPUT
fifo_datafrom[7..0]
fifo_usedw[7..0]
pad_psyncout
fifo_wrreq
fifo_rdreq
pad_dataout[7..0]
fill_state
fifo_ctrl
pad_slowclk
pad_fastclk
pad_reset
pad_psyncin
pad_dvalid
pad_datain[7..0]
fifo_datafrom[7..0]
fifo_usedw[7..0]
fill_light
pad_psyncout
fifo_wrreq
fifo_rdreq
fifo_datato[7..0]
pad_dataout[7..0]
fill_state
inst

```
-- author: kaohsiung chuah
-- date: 14/08/2003
-- filename: fifo_ctrl.vhd
-- description: this controls asynchronous fifo for zero-padding by
--              waiting for a sync byte to start
--              buffering 1 packet into fifo before reading out at
--              faster clock rate
--              when fifo is empty, reading is paused for buffering

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY fifo_ctrl IS
    PORT
    (
        pad_slowclk     : IN STD_LOGIC;
        pad_fastclk     : IN STD_LOGIC;
        pad_reset       : IN STD_LOGIC;
        pad_psyncin     : IN STD_LOGIC;
        pad_dvalid      : IN STD_LOGIC;
        pad_datain      : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        fifo_datafrom   : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        fifo_usedw      : IN UNSIGNED (7 DOWNTO 0);
        fill_light      : OUT STD_LOGIC;
        pad_psyncout    : OUT STD_LOGIC;
        fifo_wrreq      : OUT STD_LOGIC;
        fifo_rdreq      : OUT STD_LOGIC;
        fifo_datato     : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
        pad_dataout     : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
    );
END fifo_ctrl;

ARCHITECTURE rtl OF fifo_ctrl IS
    TYPE IN_STATE IS (IN_RESET, IN_GO);
    TYPE CTRL_STATE IS (CTRL_RESET, CTRL_READ, CTRL_STOP, CTRL_FILL)
;
    CONSTANT ZERO           : STD_LOGIC_VECTOR := "00000000";
    CONSTANT PAD            : INTEGER := 15;
    CONSTANT SYNC           : STD_LOGIC_VECTOR := "01000111";
    CONSTANT ISYNC          : STD_LOGIC_VECTOR := "10111000";
    CONSTANT EMPTY          : UNSIGNED := "00101111";
    CONSTANT MPEGPACK       : UNSIGNED := "10111101";
    CONSTANT MPEGPAK        : INTEGER := 188;
    SIGNAL instate          : IN_STATE;
    SIGNAL ctrlstate        : CTRL_STATE;
    SIGNAL reset_wrlatch, reset_rdlatch          : STD_LOGIC;
    SIGNAL psync_insig                           : STD_LOGIC;
    SIGNAL psync_outsig, wrreq_sig, rdreq_sig    : STD_LOGIC;
    SIGNAL data_insig, data_outsig               : STD_LOGIC_VECTOR(7
 DOWNTO 0);
    SIGNAL fifo_fromsig, fifo_tosig, usedw_sig   : STD_LOGIC_VECTOR(7
 downto 0);
BEGIN
    -- sampling input slowclk data stream and psync
    PROCESS (pad_slowclk)
    BEGIN
        IF (pad_slowclk'EVENT AND pad_slowclk = '1') THEN
            data_insig <= pad_datain;
            psync_insig <= pad_psyncin;
        END IF;
    END PROCESS;

    --sampling output fastclk data stream and psync
    PROCESS (pad_fastclk)
    BEGIN
        IF (pad_fastclk'EVENT AND pad_fastclk = '1') THEN
            pad_dataout <= data_outsig;
            fifo_fromsig <= fifo_datafrom;
        END IF;
    END PROCESS;

    -- state machine for fifo to start writing after sync byte
    PROCESS (pad_slowclk, pad_reset)
    BEGIN
        IF (pad_reset = '0') THEN
            instate <= IN_RESET;
            fifo_wrreq <= '0';
            reset_wrlatch <= '0';
        ELSIF (pad_slowclk'EVENT AND pad_slowclk = '1') THEN
            CASE instate IS
                WHEN IN_RESET =>
                    -- check if sync byte and psync in data stream
                    IF (pad_psyncin = '1' AND (pad_datain = SYNC OR
pad_datain = ISYNC)) THEN
                        -- start writing into fifo
                        instate <= IN_GO;
                        fifo_datato <= pad_datain;
                        fifo_wrreq <= '1';
                    ELSE
                        instate <= IN_RESET;
                        fifo_wrreq <= '0';
                        fifo_datato <= ZERO;
                        -- load first data to set input of fifo
                        IF (CONV_STD_LOGIC_VECTOR(fifo_usedw,8) = "0
0000000"
                            AND reset_wrlatch = '0') THEN
                            fifo_wrreq <= '1';
                            reset_wrlatch <= '1';
                        ELSE
                            fifo_wrreq <= '0';
                        END IF;
                    END IF;
                WHEN IN_GO =>
                    -- continue writting data until reset
                    IF (pad_reset = '0') THEN
                        instate <= IN_RESET;
                    ELSE
                        instate <= IN_GO;
                        -- write data into fifo
                        fifo_datato <= pad_datain;
                        IF (pad_dvalid = '0') THEN
                            fifo_wrreq <= '0';
                        ELSE
                            fifo_wrreq <= '1';
                        END IF;
                    END IF;
            END CASE;
        END IF;
    END PROCESS;

    -- fifo control
    PROCESS (pad_fastclk, pad_reset)
        VARIABLE outcount   : INTEGER RANGE 0 TO 255;
        VARIABLE stopcount  : INTEGER RANGE 0 TO 255;
    BEGIN
        -- generate psync
        -- change outcount value to sync with sync byte
        IF (outcount = 3) THEN
            pad_psyncout <= '1';
        ELSE
            pad_psyncout <= '0';
        END IF;

        -- State machine for FIFO Control
        IF (pad_reset = '0') THEN
            ctrlstate <= CTRL_RESET;
            reset_rdlatch <= '0';
            fifo_rdreq <= '0';
        ELSIF (pad_fastclk'EVENT AND pad_fastclk = '1') THEN
            CASE ctrlstate IS
                WHEN CTRL_RESET =>
                    -- check if fifo is full
                    IF (fifo_usedw = MPEGPACK) THEN
                        ctrlstate <= CTRL_READ;
                        fifo_rdreq <= '1';
                    ELSE
                        ctrlstate <= CTRL_RESET;
                        outcount := 0;
                        stopcount := 0;
                        data_outsig <= ZERO;
                        -- load first data to set output of fifo
                        IF (CONV_STD_LOGIC_VECTOR(fifo_usedw,8) = "0
0000111"
                            AND reset_rdlatch = '0') THEN
                            fifo_rdreq <= '1';
                            reset_rdlatch <= '1';
                        ELSE
                            fifo_rdreq <= '0';
                        END IF;
                    END IF;
                WHEN CTRL_READ =>
                    -- check if 188 bytes are read
                    IF (outcount = MPEGPAK) THEN
                        ctrlstate <= CTRL_STOP;
                        outcount := 0;
                        fifo_rdreq <= '0';
                        data_outsig <= fifo_datafrom;
                    ELSE
                        ctrlstate <= CTRL_READ;
                        outcount := outcount + 1;
                        data_outsig <= fifo_datafrom;
                        fifo_rdreq <= '1';
                    END IF;
                WHEN CTRL_STOP =>
                    -- check if 16 null bytes added
                    IF (stopcount = PAD) THEN
                        -- check if fifo is near empty
                        -- change empty value if required
                        IF (fifo_usedw <= EMPTY) THEN
                            ctrlstate<= CTRL_FILL;
                            outcount := outcount + 1;
                        ELSE
```

```
                                ctrlstate <= CTRL_READ;
                                stopcount := 0;
                                outcount := outcount + 1;
                                data_outsig <= ZERO;
                                fifo_rdreq <= '1';
                            END IF;
                        ELSE
                            ctrlstate <= CTRL_STOP;
                            data_outsig <= ZERO;
                            fifo_rdreq <= '0';
                            stopcount := stopcount + 1;
                            outcount := 0;
                        END IF;
                    WHEN CTRL_FILL =>
                        -- check if 188 null bytes are read
                        -- while fifo is refilled
                        IF (outcount = MPEGPAK) THEN
                            ctrlstate <= CTRL_STOP;
                            outcount := 0;
                            fifo_rdreq <= '0';
                            fill_light <= '0';
                        ELSE
                            fill_light <= '1';
                            ctrlstate <= CTRL_FILL;
                            outcount := outcount + 1;
                            stopcount := 0;
                            fifo_rdreq <= '0';
                            -- output sync byte at start of packet
                            -- output null bytes for the rest of packet
                            IF (outcount = 2) THEN
                                data_outsig <= "01000111";
                            ELSE
                                data_outsig <= "10111011";
                            END IF;
                        END IF;
                END CASE;
            END IF;
        END PROCESS;
END rtl;
```

Module: mux_ad

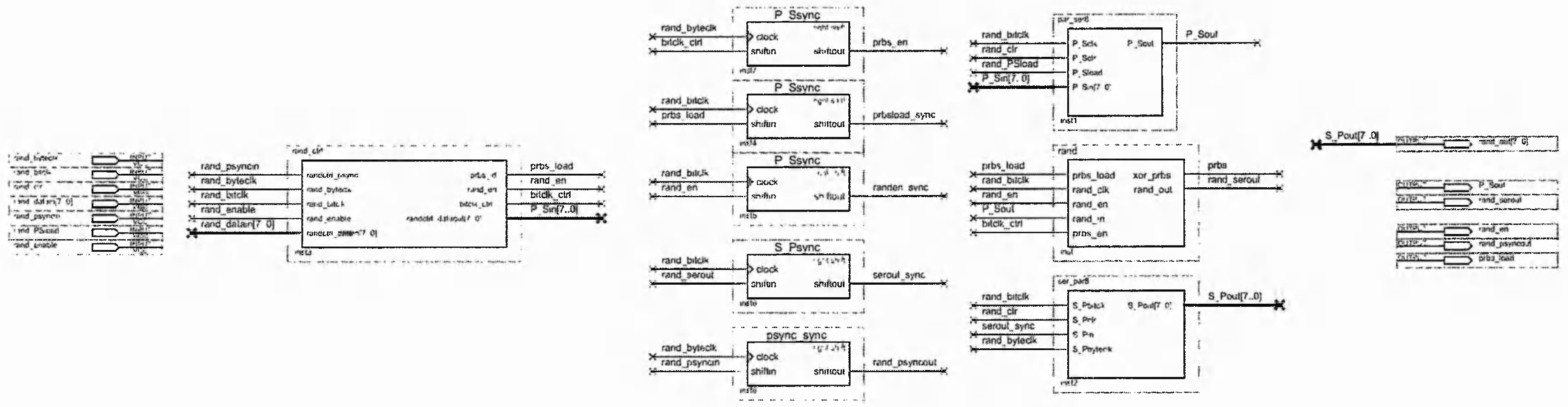
Description: SYNC byte inversion

```
-- author: kaohsiung chuah
-- date: 10/05/2002
-- filename: mux_ad.vhd
-- description: this inverts value of a sync byte after every 7
-- 31/07/2002 : input interface re-examined
-- 26/05/2004 : new simple design with enable switch.

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY mux_ad IS
    PORT
    (
        mux_clk          : IN STD_LOGIC;
        mux_psync        : IN STD_LOGIC;
        mux_reset        : IN STD_LOGIC;
        mux_en           : IN STD_LOGIC;
        mux_in           : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        mux_syncout      : OUT STD_LOGIC;
        mux_out          : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
    );
END mux_ad;

ARCHITECTURE synth OF mux_ad IS
    SIGNAL psync_sig      : STD_LOGIC;
    SIGNAL en_insig       : STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL out_sig        : STD_LOGIC_VECTOR(7 DOWNTO 0);
    CONSTANT SYNC         : STD_LOGIC_VECTOR := "01000111";
    CONSTANT ISYNC        : STD_LOGIC_VECTOR := "10111000";

BEGIN
    -- count packets, output isync after every 7 packets
    PROCESS (mux_clk, mux_en, mux_in, en_insig)
    VARIABLE pack_count : INTEGER RANGE 0 TO 7;
    BEGIN
        IF (mux_en = '0') THEN
            out_sig <= en_insig;
        ELSIF (mux_clk'EVENT AND mux_clk = '1') THEN
            IF (mux_reset = '0') THEN
                pack_count := 0;
            ELSIF (mux_psync = '1' AND mux_in = SYNC) THEN

                IF (pack_count = 0) THEN
                    -- output inverted sync
                    pack_count := pack_count + 1;
                    out_sig <= ISYNC;
                ELSE
                    pack_count := pack_count + 1;
                    out_sig <= mux_in;
                END IF;
            ELSE
                out_sig <= mux_in;
            END IF;
        END IF;
    END PROCESS;

    -- samples input and output
    PROCESS (mux_clk, mux_reset)
    BEGIN
        IF (mux_reset = '0') THEN
```

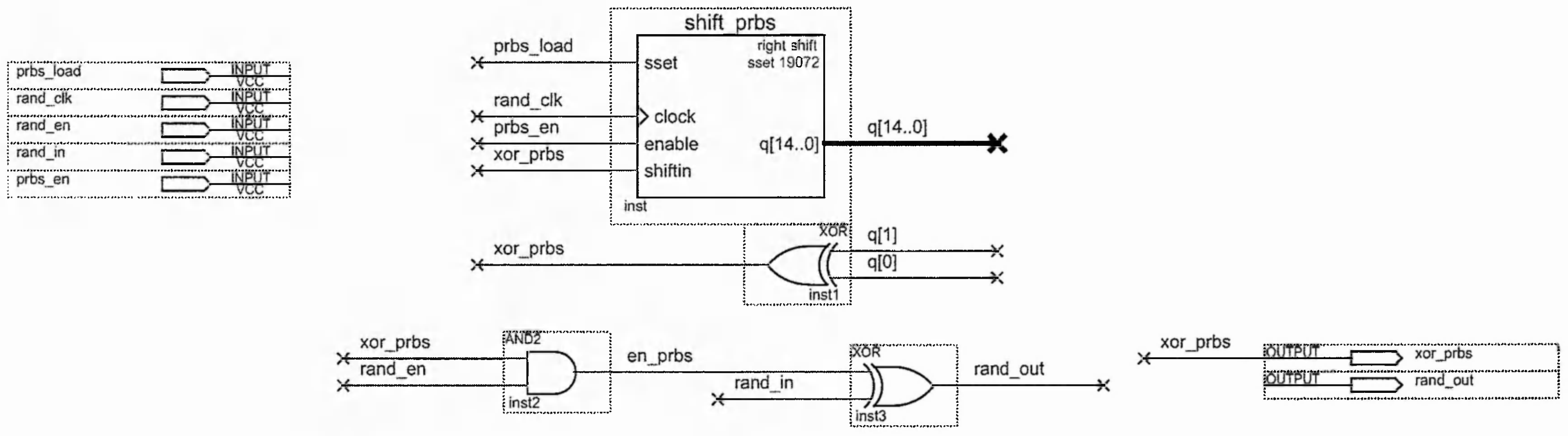
```
                mux_out <= "00000000";
            ELSIF (mux_clk'EVENT AND mux_clk = '1') THEN
                en_insig <= mux_in;
                psync_sig <= mux_psync;
                mux_out <= out_sig;
                mux_syncout <= psync_sig;
            END IF;
        END PROCESS;

    END synth;
```

Module: randomizer

Description: Randomization of MPEG packets


rand_ctrl
rand_psyncin
rand_byteclk
rand_bitclk
rand_enable
rand_datain[7..0]
prbs_load
rand_en
bitclk_ctrl
P_Sin[7..0]
inst3
P_Ssync
clock
shiftin
shiftout
prbs_en
inst7
prbsload_sync
inst4
randen_sync
inst5
S_Psync
rand_serout
serout_sync
inst6
psync_sync
rand_psyncout
inst8
par_ser8
rand_clr
rand_PSload
P_Sclk
P_Sclr
P_Sload
P_Soul
inst1
rand
prbs
xor_prbs
rand_out
rand_clk
P_Soul
inst
ser_par8
S_Psclk
S_Pclr
S_Pen
S_Pbyteclk
S_Pout[7..0]
inst2
rand_out[7..0]

Date: March 25, 2006 ../randomizer/rand.bdf* Project: dvbenc_burn



Page 1 of 1 Revision: dvbenc_burn

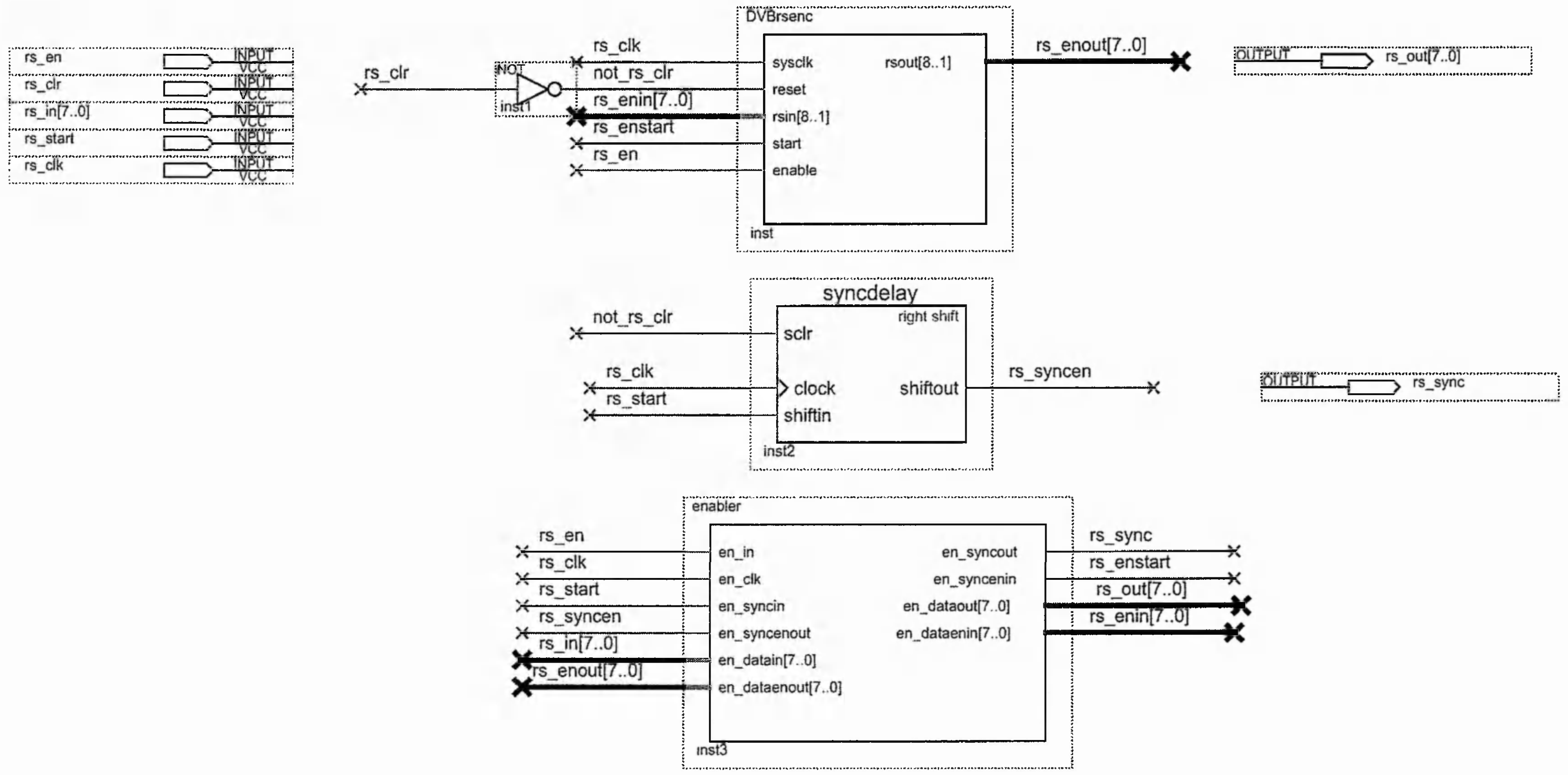
```vhdl
-- author: kaohsiung chuah
-- date: 17/11/2003
-- filename: randomizer.vhd
-- description: this controls the workings of randomiser
--              prbs_ld reloads the prbs generator
--              rand_en for on/off randonisation
--              bitclk_ctrl start/pause prbs generation
-- Revised 15/06/2004: rand_enable added.

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY rand_ctrl IS
    PORT
    (
        randctrl_psync      : IN STD_LOGIC;
        rand_byteclk        : IN STD_LOGIC;
        rand_bitclk         : IN STD_LOGIC;
        rand_enable         : IN STD_LOGIC;
        randctrl_datain     : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        prbs_ld             : OUT STD_LOGIC;
        rand_en             : OUT STD_LOGIC;
        bitclk_ctrl         : OUT STD_LOGIC;
        randctrl_dataout    : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
    );
END rand_ctrl;

ARCHITECTURE synth OF rand_ctrl IS
    CONSTANT ISYNC          : std_logic_vector := "10111000";
    CONSTANT SYNC           : std_logic_vector := "01000111";
    SIGNAL datain_sig       : STD_LOGIC_VECTOR (7 DOWNTO 0);
    SIGNAL psync_sig        : STD_LOGIC;
    SIGNAL randctrl_state   : STD_LOGIC_VECTOR (1 DOWNTO 0);

BEGIN
    -- sampling data and psync signal at rising edge
    PROCESS (rand_byteclk)
    BEGIN
        IF (rand_byteclk'EVENT AND rand_byteclk = '1') THEN
            datain_sig <= randctrl_datain;
            psync_sig <= randctrl_psync;
        END IF;
    END PROCESS;

    -- control for randomiser
    PROCESS (rand_byteclk, rand_enable)
    BEGIN
        randctrl_dataout <= datain_sig;
        IF (rand_enable = '0') THEN
            prbs_ld <= '0';
            rand_en <= '0';
        ELSIF (rand_byteclk'EVENT AND rand_byteclk = '1') THEN
            IF (datain_sig = ISYNC AND psync_sig = '1') THEN
                prbs_ld <= '1';
                rand_en <= '0';
            ELSIF (datain_sig = SYNC AND psync_sig = '1') THEN
                prbs_ld <= '0';
                rand_en <= '0';
            ELSE
                prbs_ld <= '0';
                rand_en <= '1';
            END IF;
        END IF;
    END PROCESS;

    -- additional control to accomodate additional 16 rs-bytes
    PROCESS (rand_bitclk)
        VARIABLE prbs_count      : INTEGER RANGE 0 TO 2047;
    BEGIN
        IF (rand_bitclk'EVENT AND rand_bitclk = '1') THEN
            IF (psync_sig = '1') THEN
                prbs_count := 0;
                bitclk_ctrl <= '1';
            ELSIF (prbs_count = 1496) THEN
                -- pause randomisation during 16 rs-bytes
                bitclk_ctrl <= '0';
                prbs_count := 0;
            ELSE
                prbs_count := prbs_count + 1;
            END IF;
        END IF;
    END PROCESS;
END synth;
```

```
-- author: kaohsiung chuah
-- date: 24/03/2003
-- filename: par_ser8.vhd
-- description: this is a 8-bit parallel-serial converter
--              P_Sload is 1 to load codeword

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY par_ser8 IS
    PORT
    (
        P_Sclk      : IN STD_LOGIC;
        P_Sclr      : IN STD_LOGIC;
        P_Sload     : IN STD_LOGIC;
        P_Sin       : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        P_Sout      : OUT STD_LOGIC
    );
END par_ser8;

ARCHITECTURE rtl OF par_ser8 IS
    SIGNAL in_sig   : STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
    PROCESS (P_Sclk, in_sig, P_Sclr)
    BEGIN
        P_Sout <= in_sig(7);
        IF (P_Sclr = '0') THEN
            in_sig <= "00000000";
        ELSE
            IF (P_Sclk'EVENT AND P_Sclk = '1') THEN
                IF (P_Sload = '1') THEN
                    -- load codeword
                    in_sig <= P_Sin;
                ELSE
                    in_sig(7 downto 1) <= in_sig(6 downto 0);
                END IF;
            END IF;
        END IF;
    END PROCESS;
END rtl;
```

```
-- author: kaohsiung chuah
-- date: 21/05/2003
-- filename: ser_par8.vhd
-- description: this is a 8-bit serial-parallel converter
--              S_Pbitclk connects to bit-rate clock
--              A_Pbyteclk connects to byte-rate clock

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ser_par8 IS
    PORT
    (
        S_Pbitclk   : IN STD_LOGIC;
        S_Pclr      : IN STD_LOGIC;
        S_Pin       : IN STD_LOGIC;
        S_Pbyteclk  : IN STD_LOGIC;
        S_Pout      : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
    );
END ser_par8;

ARCHITECTURE rtl OF ser_par8 IS
    SIGNAL in_sig       : STD_LOGIC;
    SIGNAL outsig       : STD_LOGIC_VECTOR(6 DOWNTO 0);
    SIGNAL S_Poutsig    : STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
    PROCESS (S_Pbitclk, S_Pbyteclk, S_Pclr)
    BEGIN
        IF (S_Pclr = '0') THEN
            outsig <= "0000000";
        ELSIF (S_Pbitclk'EVENT AND S_Pbitclk = '1') THEN
                in_sig <= S_Pin;
                -- shift in serial bit
                outsig(0) <= in_sig;
                outsig(6 downto 1) <= outsig(5 downto 0);
        END IF;
        IF (S_Pbyteclk'EVENT AND S_Pbyteclk = '1') THEN
                -- output codeword
                S_Pout(7 downto 1) <= outsig;
                S_Pout(0) <= in_sig;
        END IF;
    END PROCESS;
END rtl;
```

Module: rsenc

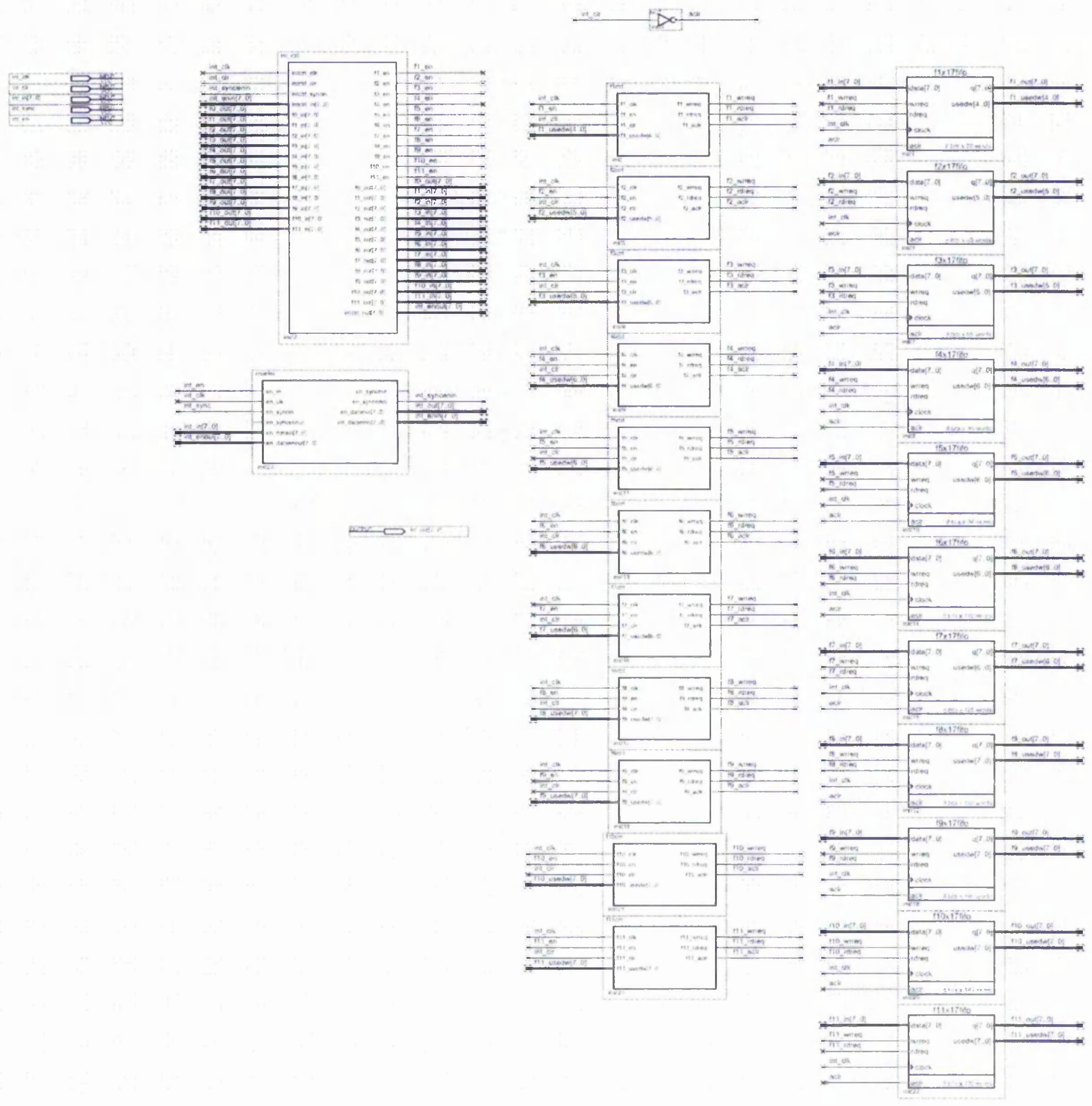
Description: Reed-Solomon (204,188) encoding


rs_en
rs_clr
rs_in[7..0]
rs_start
rs_clk
INPUT
VCC
rs_clr
NOT
inst1
rs_clk
not_rs_clr
rs_enin[7..0]
rs_enstart
rs_en
DVBrsenc
sysclk
reset
rsin[8..1]
start
enable
rsout[8..1]
inst
rs_enout[7..0]
OUTPUT
rs_out[7..0]
syncdelay
right shift
not_rs_clr
sclr
rs_clk
clock
rs_start
shiftin
shiftout
inst2
rs_syncen
OUTPUT
rs_sync
enabler
rs_en
rs_clk
rs_start
rs_syncen
rs_in[7..0]
rs_enout[7..0]
en_in
en_clk
en_syncin
en_syncenout
en_datain[7..0]
en_dataenout[7..0]
en_syncout
en_syncenin
en_dataout[7..0]
en_dataenin[7..0]
inst3
rs_sync
rs_enstart
rs_out[7..0]
rs_enin[7..0]

```
-- author: kaohsiung chuah
-- date: 24/06/2004
-- filename: enabler.vhd
-- description: this enables a functional module to switch on/off
--              when en_in is 0 module is bypassed.
--              when en_in is 1 module is in operation.
--              en_syncin,en_datain input codeword
--              en_syncenin,en_dataenin input codeword to module
--              en_syncout,en_dataout output cordword
--              en_syncenout,en_dataenout output processed codeword
--              from module

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY enabler IS
    PORT
    (
        en_in           : IN STD_LOGIC;
        en_clk          : IN STD_LOGIC;
        en_syncin       : IN STD_LOGIC;
        en_syncenout    : IN STD_LOGIC;
        en_datain       : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        en_dataenout    : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        en_syncout      : OUT STD_LOGIC;
        en_syncenin     : OUT STD_LOGIC;
        en_dataout      : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
        en_dataenin     : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
    );
END enabler;

ARCHITECTURE synth OF enabler IS
BEGIN
    PROCESS (en_clk, en_in)
    BEGIN
        IF (en_clk'EVENT AND en_clk = '1') THEN
            IF (en_in = '0') THEN
                -- bypass process, direct input/ouput
                en_dataout <= en_datain;
                en_syncout <= en_syncin;
                en_dataenin <= "00000000";
                en_syncenin <= '0';
            ELSE
                -- enable process, input/output to/from module
                en_dataout <= en_dataenout;
                en_syncout <= en_syncenout;
                en_dataenin <= en_datain;
                en_syncenin <= en_syncin;
            END IF;
        END IF;
    END PROCESS;
END synth;
```

Module: interleaver

Description: Convolutional interleaving



```
-- author: kaohsiung chuah
-- date: 29/03/2004
-- filename: int_ctrl.vhd
-- description: this controls the interleaver by
--              intctrl_in inputs the codewords
--              f0_in -> f11_in inputs codewords from fifos
--              f1_en -> f11_en enables the fifos
--              f1_out -> f11_out outputs codewords to fifos
--              intctrl_out outputs interleaved codewords
--              state machine waits for psync to initiate
--              state machine changes states in at every clock cycle
--              at each state codewords are sent to and from fifo

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY int_ctrl IS
    PORT
    (
        intctrl_clk     : IN STD_LOGIC;
        intctrl_clr     : IN STD_LOGIC;
        intctrl_syncin  : IN STD_LOGIC;
        intctrl_in      : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        f0_in           : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        f1_in           : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        f2_in           : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        f3_in           : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        f4_in           : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        f5_in           : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        f6_in           : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        f7_in           : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        f8_in           : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        f9_in           : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        f10_in          : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        f11_in          : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        f1_en           : OUT STD_LOGIC;
        f2_en           : OUT STD_LOGIC;
        f3_en           : OUT STD_LOGIC;
        f4_en           : OUT STD_LOGIC;
        f5_en           : OUT STD_LOGIC;
        f6_en           : OUT STD_LOGIC;
        f7_en           : OUT STD_LOGIC;
        f8_en           : OUT STD_LOGIC;
        f9_en           : OUT STD_LOGIC;
        f10_en          : OUT STD_LOGIC;
        f11_en          : OUT STD_LOGIC;
        f0_out          : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
        f1_out          : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
        f2_out          : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
        f3_out          : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
        f4_out          : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
        f5_out          : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
        f6_out          : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
        f7_out          : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
        f8_out          : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
        f9_out          : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
        f10_out         : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
        f11_out         : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
        intctrl_out     : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
    );
END int_ctrl;
```

```
ARCHITECTURE synth OF int_ctrl IS
    TYPE STATE_TYPE IS (RESET, INIT, ZERO, ONE, TWO, THREE, FOUR, FI
VE,
                        SIX, SEVEN, EIGHT, NINE, TEN, ELEVEN);
    SIGNAL intstate     : STATE_TYPE;
    SIGNAL in_sig       : STD_LOGIC_VECTOR (7 DOWNTO 0);
    SIGNAL get_sig      : STD_LOGIC_VECTOR (7 DOWNTO 0);
    SIGNAL get_sync     : STD_LOGIC;
    SIGNAL clrlatch     : STD_LOGIC;
BEGIN
    -- Sampling input word
    PROCESS (intctrl_clk, intctrl_clr)
    BEGIN
        IF (intctrl_clr = '0') THEN
            get_sig <= "00000000";
        ELSIF (intctrl_clk'EVENT AND intctrl_clk = '1') THEN
            get_sync <= intctrl_syncin;
            get_sig <= intctrl_in;
            in_sig <= get_sig;
        END IF;
    END PROCESS;

    -- State machine for Interleaver Control
    -- 12 states for 12 branch + reset and initiate
    PROCESS (intctrl_clk, intctrl_clr)
    BEGIN
        IF (intctrl_clr = '0') THEN
            intstate <= RESET;
            clrlatch <= '0';
        ELSIF (intctrl_clk'EVENT and intctrl_clk = '1') THEN
            CASE intstate IS
                WHEN RESET =>
                    intstate <= INIT;
                WHEN INIT =>
                    IF (get_sync = '1' AND (get_sig = "01000111" OR
get_sig = "10111000")) THEN
                        intstate <= ZERO;
                    ELSE
                        intstate <= INIT;
                        IF (clrlatch = '0') THEN
                            clrlatch <= '1';
                        END IF;
                    END IF;
                WHEN ZERO =>
                    intstate <= ONE;
                WHEN ONE =>
                    intstate <= TWO;
                WHEN TWO =>
                    intstate <= THREE;
                WHEN THREE =>
                    intstate <= FOUR;
                WHEN FOUR =>
                    intstate <= FIVE;
                WHEN FIVE =>
                    intstate <= SIX;
                WHEN SIX =>
                    intstate <= SEVEN;
                WHEN SEVEN =>
                    intstate <= EIGHT;
                WHEN EIGHT =>
```

```
                        intstate <= NINE;
                    WHEN NINE =>
                        intstate <= TEN;
                    WHEN TEN =>
                        intstate <= ELEVEN;
                    WHEN ELEVEN =>
                        intstate <= ZERO;
                END CASE;
            END IF;
        END PROCESS;

        -- send enable signal to corresponding fifo
        f1_en <= '1' WHEN (intstate = ONE) ELSE '0';
        f2_en <= '1' WHEN (intstate = TWO) ELSE '0';
        f3_en <= '1' WHEN intstate = THREE ELSE '0';
        f4_en <= '1' WHEN intstate = FOUR ELSE '0';
        f5_en <= '1' WHEN intstate = FIVE ELSE '0';
        f6_en <= '1' WHEN intstate = SIX ELSE '0';
        f7_en <= '1' WHEN intstate = SEVEN ELSE '0';
        f8_en <= '1' WHEN intstate = EIGHT ELSE '0';
        f9_en <= '1' WHEN intstate = NINE ELSE '0';
        f10_en <= '1' WHEN intstate = TEN ELSE '0';
        f11_en <= '1' WHEN intstate = ELEVEN ELSE '0';

        -- send input codeword to corresponding fifo
        f0_out <= in_sig WHEN intstate = ZERO ELSE "00000000";
        f1_out <= in_sig WHEN intstate = ONE ELSE "00000000";
        f2_out <= in_sig WHEN intstate = TWO ELSE "00000000";
        f3_out <= in_sig WHEN intstate = THREE ELSE "00000000";
        f4_out <= in_sig WHEN intstate = FOUR ELSE "00000000";
        f5_out <= in_sig WHEN intstate = FIVE ELSE "00000000";
        f6_out <= in_sig WHEN intstate = SIX ELSE "00000000";
        f7_out <= in_sig WHEN intstate = SEVEN ELSE "00000000";
        f8_out <= in_sig WHEN intstate = EIGHT ELSE "00000000";
        f9_out <= in_sig WHEN intstate = NINE ELSE "00000000";
        f10_out <= in_sig WHEN intstate = TEN ELSE "00000000";
        f11_out <= in_sig WHEN intstate = ELEVEN ELSE "00000000";

        -- interleaved output from fifos at corresponding state
        WITH intstate SELECT
            intctrl_out <=  f0_in WHEN ZERO,
                            f1_in WHEN ONE,
                            f2_in WHEN TWO,
                            f3_in WHEN THREE,
                            f4_in WHEN FOUR,
                            f5_in WHEN FIVE,
                            f6_in WHEN SIX,
                            f7_in WHEN SEVEN,
                            f8_in WHEN EIGHT,
                            f9_in WHEN NINE,
                            f10_in WHEN TEN,
                            f11_in WHEN ELEVEN,
                            "00000000" WHEN OTHERS;

    END synth;
```

```
-- author: kaohsiung chuah
-- date: 01/04/2004
-- filename: f1ctrl.vhd
-- description: this controls fifo to perform as shift register
--              by buffering fifo until required length
--              then allow fifo to read and write at the same time
--              to function as shift register

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY f1ctrl IS
    PORT
    (
        f1_clk      : IN STD_LOGIC;
        f1_en       : IN STD_LOGIC;
        f1_clr      : IN STD_LOGIC;
        f1_usedw    : IN UNSIGNED (4 DOWNTO 0);
        f1_wrreq    : OUT STD_LOGIC;
        f1_rdreq    : OUT STD_LOGIC;
        f1_aclr     : OUT STD_LOGIC
    );
END f1ctrl;

ARCHITECTURE synth OF f1ctrl IS
BEGIN
    PROCESS (f1_clr, f1_en, f1_usedw)
    BEGIN
        IF (f1_clr = '0') THEN
            f1_aclr <= '1';
            f1_rdreq <= '0';
            f1_wrreq <= '0';
        ELSIF (CONV_STD_LOGIC_VECTOR(f1_usedw,5) >= "10000") THEN
            -- perform as shift register
            f1_rdreq <= f1_en;
            f1_wrreq <= f1_en;
            f1_aclr <= '0';
        ELSE
            -- buffering fifo
            f1_wrreq <= f1_en;
            f1_rdreq <= '0';
            f1_aclr <= '0';
        END IF;
    END PROCESS;
END synth;
```

```
-- author: kaohsiung chuah
-- date: 01/04/2004
-- filename: f2ctrl.vhd
-- description: this controls fifo to perform as shift register
--              by buffering fifo until required length
--              then allow fifo to read and write at the same time
--              to function as shift register

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY f2ctrl IS
    PORT
    (
        f2_clk      : IN STD_LOGIC;
        f2_en       : IN STD_LOGIC;
        f2_clr      : IN STD_LOGIC;
        f2_usedw    : IN UNSIGNED (5 DOWNTO 0);
        f2_wrreq    : OUT STD_LOGIC;
        f2_rdreq    : OUT STD_LOGIC;
        f2_aclr     : OUT STD_LOGIC
    );
END f2ctrl;

ARCHITECTURE synth OF f2ctrl IS
BEGIN
    PROCESS (f2_clr, f2_en, f2_usedw)
    BEGIN
        IF (f2_clr = '0') THEN
            f2_aclr <= '1';
            f2_rdreq <= '0';
            f2_wrreq <= '0';
        ELSIF (CONV_STD_LOGIC_VECTOR(f2_usedw,6) >= "100001") THEN
            -- perform as shift register
            f2_rdreq <= f2_en;
            f2_wrreq <= f2_en;
            f2_aclr <= '0';
        ELSE
            -- buffering fifo
            f2_wrreq <= f2_en;
            f2_rdreq <= '0';
            f2_aclr <= '0';
        END IF;
    END PROCESS;
END synth;
```

```
-- author: kaohsiung chuah
-- date: 01/04/2004
-- filename: f3ctrl.vhd
-- description: this controls fifo to perform as shift register
--              by buffering fifo until required length
--              then allow fifo to read and write at the same time
--              to function as shift register

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY f3ctrl IS
    PORT
    (
        f3_clk      : IN STD_LOGIC;
        f3_en       : IN STD_LOGIC;
        f3_clr      : IN STD_LOGIC;
        f3_usedw    : IN UNSIGNED (5 DOWNTO 0);
        f3_wrreq    : OUT STD_LOGIC;
        f3_rdreq    : OUT STD_LOGIC;
        f3_aclr     : OUT STD_LOGIC
    );
END f3ctrl;

ARCHITECTURE synth OF f3ctrl IS
BEGIN
    PROCESS (f3_clr, f3_en, f3_usedw)
    BEGIN
        IF (f3_clr = '0') THEN
            f3_aclr <= '1';
            f3_rdreq <= '0';
            f3_wrreq <= '0';
        ELSIF (CONV_STD_LOGIC_VECTOR(f3_usedw,6) >= "110010") THEN
            -- perform as shift register
            f3_rdreq <= f3_en;
            f3_wrreq <= f3_en;
            f3_aclr <= '0';
        ELSE
            -- buffering fifo
            f3_wrreq <= f3_en;
            f3_rdreq <= '0';
            f3_aclr <= '0';
        END IF;
    END PROCESS;
END synth;
```

```
-- author: kaohsiung chuah
-- date: 01/04/2004
-- filename: f4ctrl.vhd
-- description: this controls fifo to perform as shift register
--              by buffering fifo until required length
--              then allow fifo to read and write at the same time
--              to function as shift register

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY f4ctrl IS
    PORT
    (
        f4_clk      : IN STD_LOGIC;
        f4_en       : IN STD_LOGIC;
        f4_clr      : IN STD_LOGIC;
        f4_usedw    : IN UNSIGNED (6 DOWNTO 0);
        f4_wrreq    : OUT STD_LOGIC;
        f4_rdreq    : OUT STD_LOGIC;
        f4_aclr     : OUT STD_LOGIC
    );
END f4ctrl;

ARCHITECTURE synth OF f4ctrl IS
BEGIN
    PROCESS (f4_clr, f4_en, f4_usedw)
    BEGIN
        IF (f4_clr = '0') THEN
            f4_aclr <= '1';
            f4_rdreq <= '0';
            f4_wrreq <= '0';
        ELSIF (CONV_STD_LOGIC_VECTOR(f4_usedw,7) >= "1000011") THEN
            -- perform as shift register
            f4_rdreq <= f4_en;
            f4_wrreq <= f4_en;
            f4_aclr <= '0';
        ELSE
            -- buffering fifo
            f4_wrreq <= f4_en;
            f4_rdreq <= '0';
            f4_aclr <= '0';
        END IF;
    END PROCESS;
END synth;
```

```
-- author: kaohsiung chuah
-- date: 01/04/2004
-- filename: f5ctrl.vhd
-- description: this controls fifo to perform as shift register
--              by buffering fifo until required length
--              then allow fifo to read and write at the same time
--              to function as shift register

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY f5ctrl IS
    PORT
    (
        f5_clk      : IN STD_LOGIC;
        f5_en       : IN STD_LOGIC;
        f5_clr      : IN STD_LOGIC;
        f5_usedw    : IN UNSIGNED (6 DOWNTO 0);
        f5_wrreq    : OUT STD_LOGIC;
        f5_rdreq    : OUT STD_LOGIC;
        f5_aclr     : OUT STD_LOGIC
    );
END f5ctrl;

ARCHITECTURE synth OF f5ctrl IS
BEGIN
    PROCESS (f5_clr, f5_en, f5_usedw)
    BEGIN
        IF (f5_clr = '0') THEN
            f5_aclr <= '1';
            f5_rdreq <= '0';
            f5_wrreq <= '0';
        ELSIF (CONV_STD_LOGIC_VECTOR(f5_usedw,7) >= "1010100") THEN
            -- perform as shift register
            f5_rdreq <= f5_en;
            f5_wrreq <= f5_en;
            f5_aclr <= '0';
        ELSE
            -- buffering fifo
            f5_wrreq <= f5_en;
            f5_rdreq <= '0';
            f5_aclr <= '0';
        END IF;
    END PROCESS;
END synth;
```

```
-- author: kaohsiung chuah
-- date: 01/04/2004
-- filename: f6ctrl.vhd
-- description: this controls fifo to perform as shift register
--              by buffering fifo until required length
--              then allow fifo to read and write at the same time
--              to function as shift register

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY f6ctrl IS
    PORT
    (
        f6_clk      : IN STD_LOGIC;
        f6_en       : IN STD_LOGIC;
        f6_clr      : IN STD_LOGIC;
        f6_usedw    : IN UNSIGNED (6 DOWNTO 0);
        f6_wrreq    : OUT STD_LOGIC;
        f6_rdreq    : OUT STD_LOGIC;
        f6_aclr     : OUT STD_LOGIC
    );
END f6ctrl;

ARCHITECTURE synth OF f6ctrl IS
BEGIN
    PROCESS (f6_clr, f6_en, f6_usedw)
    BEGIN
        IF (f6_clr = '0') THEN
            f6_aclr <= '1';
            f6_rdreq <= '0';
            f6_wrreq <= '0';
        ELSIF (CONV_STD_LOGIC_VECTOR(f6_usedw,7) >= "1100101") THEN
            -- perform as shift register
            f6_rdreq <= f6_en;
            f6_wrreq <= f6_en;
            f6_aclr <= '0';
        ELSE
            -- buffering fifo
            f6_wrreq <= f6_en;
            f6_rdreq <= '0';
            f6_aclr <= '0';
        END IF;
    END PROCESS;
END synth;
```

```
-- author: kaohsiung chuah
-- date: 01/04/2004
-- filename: f7ctrl.vhd
-- description: this controls fifo to perform as shift register
--              by buffering fifo until required length
--              then allow fifo to read and write at the same time
--              to function as shift register

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY f7ctrl IS
    PORT
    (
        f7_clk      : IN STD_LOGIC;
        f7_en       : IN STD_LOGIC;
        f7_clr      : IN STD_LOGIC;
        f7_usedw    : IN UNSIGNED (6 DOWNTO 0);
        f7_wrreq    : OUT STD_LOGIC;
        f7_rdreq    : OUT STD_LOGIC;
        f7_aclr     : OUT STD_LOGIC
    );
END f7ctrl;

ARCHITECTURE synth OF f7ctrl IS
BEGIN
    PROCESS (f7_clr, f7_en, f7_usedw)
    BEGIN
        IF (f7_clr = '0') THEN
            f7_aclr <= '1';
            f7_rdreq <= '0';
            f7_wrreq <= '0';
        ELSIF (CONV_STD_LOGIC_VECTOR(f7_usedw,7) >= "1110110") THEN
            -- perform as shift register
            f7_rdreq <= f7_en;
            f7_wrreq <= f7_en;
            f7_aclr <= '0';
        ELSE
            -- buffering fifo
            f7_wrreq <= f7_en;
            f7_rdreq <= '0';
            f7_aclr <= '0';
        END IF;
    END PROCESS;
END synth;
```

```
-- author: kaohsiung chuah
-- date: 01/04/2004
-- filename: f8ctrl.vhd
-- description: this controls fifo to perform as shift register
--              by buffering fifo until required length
--              then allow fifo to read and write at the same time
--              to function as shift register

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY f8ctrl IS
    PORT
    (
        f8_clk      : IN STD_LOGIC;
        f8_en       : IN STD_LOGIC;
        f8_clr      : IN STD_LOGIC;
        f8_usedw    : IN UNSIGNED (7 DOWNTO 0);
        f8_wrreq    : OUT STD_LOGIC;
        f8_rdreq    : OUT STD_LOGIC;
        f8_aclr     : OUT STD_LOGIC
    );
END f8ctrl;

ARCHITECTURE synth OF f8ctrl IS
BEGIN
    PROCESS (f8_clr, f8_en, f8_usedw)
    BEGIN
        IF (f8_clr = '0') THEN
            f8_aclr <= '1';
            f8_rdreq <= '0';
            f8_wrreq <= '0';
        ELSIF (CONV_STD_LOGIC_VECTOR(f8_usedw,8) >= "10000111") THEN
            -- performing as shift register
            f8_rdreq <= f8_en;
            f8_wrreq <= f8_en;
            f8_aclr <= '0';
        ELSE
            -- buffering fifo
            f8_wrreq <= f8_en;
            f8_rdreq <= '0';
            f8_aclr <= '0';
        END IF;
    END PROCESS;
END synth;
```

```
-- author: kaohsiung chuah
-- date: 01/04/2004
-- filename: f9ctrl.vhd
-- description: this controls fifo to perform as shift register
--              by buffering fifo until required length
--              then allow fifo to read and write at the same time
--              to function as shift register

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY f9ctrl IS
    PORT
    (
        f9_clk      : IN STD_LOGIC;
        f9_en       : IN STD_LOGIC;
        f9_clr      : IN STD_LOGIC;
        f9_usedw    : IN UNSIGNED (7 DOWNTO 0);
        f9_wrreq    : OUT STD_LOGIC;
        f9_rdreq    : OUT STD_LOGIC;
        f9_aclr     : OUT STD_LOGIC
    );
END f9ctrl;

ARCHITECTURE synth OF f9ctrl IS
BEGIN
    PROCESS (f9_clr, f9_en, f9_usedw)
    BEGIN
        IF (f9_clr = '0') THEN
            f9_aclr <= '1';
            f9_rdreq <= '0';
            f9_wrreq <= '0';
        ELSIF (CONV_STD_LOGIC_VECTOR(f9_usedw,8) >= "10011000") THEN
            -- performing as shift register
            f9_rdreq <= f9_en;
            f9_wrreq <= f9_en;
            f9_aclr <= '0';
        ELSE
            -- buffering fifo
            f9_wrreq <= f9_en;
            f9_rdreq <= '0';
            f9_aclr <= '0';
        END IF;
    END PROCESS;
END synth;
```

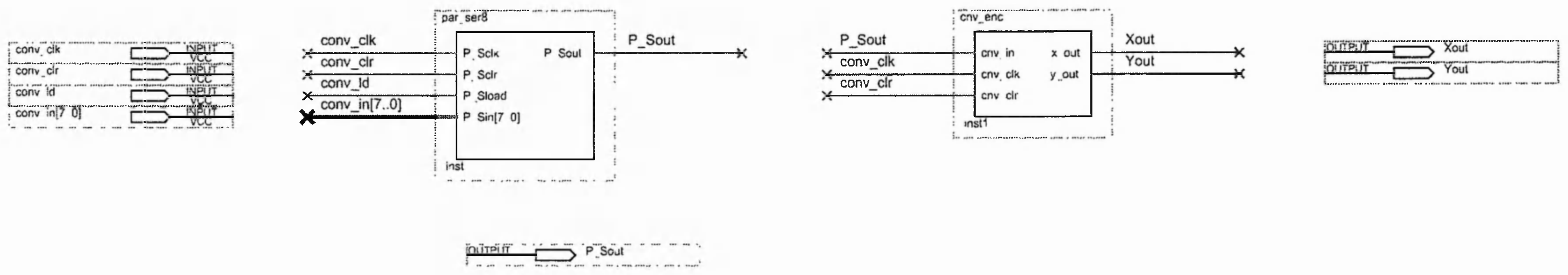
```
-- author: kaohsiung chuah
-- date: 01/04/2004
-- filename: f10ctrl.vhd
-- description: this controls fifo to perform as shift register
--              by buffering fifo until required length
--              then allow fifo to read and write at the same time
--              to function as shift register

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY f10ctrl IS
    PORT
    (
        f10_clk     : IN STD_LOGIC;
        f10_en      : IN STD_LOGIC;
        f10_clr     : IN STD_LOGIC;
        f10_usedw   : IN UNSIGNED (7 DOWNTO 0);
        f10_wrreq   : OUT STD_LOGIC;
        f10_rdreq   : OUT STD_LOGIC;
        f10_aclr        : OUT STD_LOGIC
    );
END f10ctrl;

ARCHITECTURE synth OF f10ctrl IS
BEGIN
    PROCESS (f10_clr, f10_en, f10_usedw)
    BEGIN
        IF (f10_clr = '0') THEN
            f10_aclr <= '1';
            f10_rdreq <= '0';
            f10_wrreq <= '0';
        ELSIF (CONV_STD_LOGIC_VECTOR(f10_usedw,8) >= "10101001") THE
N
            -- performing as shift register
            f10_rdreq <= f10_en;
            f10_wrreq <= f10_en;
            f10_aclr <= '0';
        ELSE
            -- buffering fifo
            f10_wrreq <= f10_en;
            f10_rdreq <= '0';
            f10_aclr <= '0';
        END IF;
    END PROCESS;
END synth;
```

```
-- author: kaohsiung chuah
-- date: 01/04/2004
-- filename: f11ctrl.vhd
-- description: this controls fifo to perform as shift register
--              by buffering fifo until required length
--              then allow fifo to read and write at the same time
--              to function as shift register

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY f11ctrl IS
    PORT
    (
        f11_clk     : IN STD_LOGIC;
        f11_en      : IN STD_LOGIC;
        f11_clr     : IN STD_LOGIC;
        f11_usedw   : IN UNSIGNED (7 DOWNTO 0);
        f11_wrreq   : OUT STD_LOGIC;
        f11_rdreq   : OUT STD_LOGIC;
        f11_aclr        : OUT STD_LOGIC
    );
END f11ctrl;

ARCHITECTURE synth OF f11ctrl IS
BEGIN
    PROCESS (f11_clr, f11_en, f11_usedw)
    BEGIN
        IF (f11_clr = '0') THEN
            f11_aclr <= '1';
            f11_rdreq <= '0';
            f11_wrreq <= '0';
        ELSIF (CONV_STD_LOGIC_VECTOR(f11_usedw,8) >= "10111010") THE
N
            -- performing as shift register
            f11_rdreq <= f11_en;
            f11_wrreq <= f11_en;
            f11_aclr <= '0';
        ELSE
            -- buffering fifo
            f11_wrreq <= f11_en;
            f11_rdreq <= '0';
            f11_aclr <= '0';
        END IF;
    END PROCESS;
END synth;
```

```
-- author: kaohsiung chuah
-- date: 24/06/2004
-- filename: enabler.vhd
-- description: this enables a functional module to switch on/off
--              when en_in is 0 module is bypassed.
--              when en_in is 1 module is in operation.
--              en_syncin,en_datain input codeword
--              en_syncenin,en_dataenin input codeword to module
--              en_syncout,en_dataout output cordword
--              en_syncenout,en_dataenout output processed codeword
--              from module

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY enabler IS
    PORT
    (
        en_in           : IN STD_LOGIC;
        en_clk          : IN STD_LOGIC;
        en_syncin       : IN STD_LOGIC;
        en_syncenout    : IN STD_LOGIC;
        en_datain       : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        en_dataenout    : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        en_syncout      : OUT STD_LOGIC;
        en_syncenin     : OUT STD_LOGIC;
        en_dataout      : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
        en_dataenin     : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
    );
END enabler;

ARCHITECTURE synth OF enabler IS
BEGIN
    PROCESS (en_clk, en_in)
    BEGIN
        IF (en_clk'EVENT AND en_clk = '1') THEN
            IF (en_in = '0') THEN
                -- bypass process, direct input/ouput
                en_dataout <= en_datain;
                en_syncout <= en_syncin;
                en_dataenin <= "00000000";
                en_syncenin <= '0';
            ELSE
                -- enable process, input/output to/from module
                en_dataout <= en_dataenout;
                en_syncout <= en_syncenout;
                en_dataenin <= en_datain;
                en_syncenin <= en_syncin;
            END IF;
        END IF;
    END PROCESS;
END synth;
```

Module: conv_enc

Description: Convolutional encoding

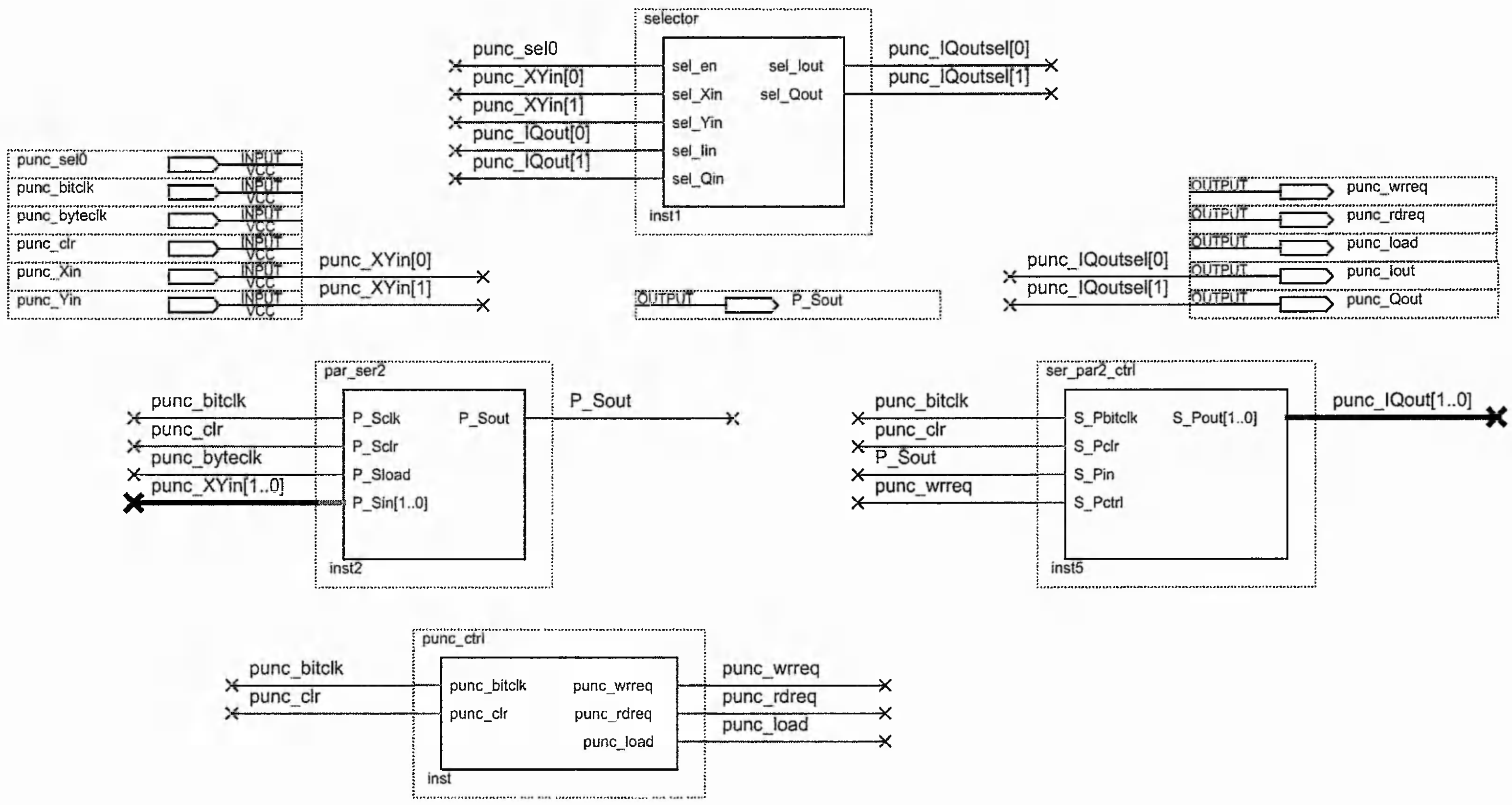

conv_clk
conv_clr
conv_ld
conv_in[7..0]
INPUT
VCC
par_ser8
P_Sclk
P_Sclr
P_Sload
P_Sin[7..0]
P_Sout
inst
OUTPUT
P_Sout
conv_clk
conv_clr
cnv_enc
cnv_in
cnv_clk
cnv_clr
x_out
y_out
inst1
Xout
Yout
OUTPUT
Xout
Yout

```
-- author: kaohsiung chuah
-- date: 25/03/2003
-- filename: cnv_enc.vhd
-- description: this is a convolutional encoder
--              cnv_in shifts input bits
--              x_out and y_out are X and Y encoded output


LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY cnv_enc IS
    PORT
    (
        cnv_in      : IN std_logic;
        cnv_clk     : IN std_logic;
        cnv_clr     : IN std_logic;
        x_out       : OUT std_logic;
        y_out       : OUT std_logic
    );
END cnv_enc;

ARCHITECTURE rtl OF cnv_enc IS
    SIGNAL in_latch         : STD_LOGIC_VECTOR(6 DOWNTO 0);

BEGIN
    PROCESS (cnv_clk, cnv_clr)
    BEGIN
        IF (cnv_clr = '0') THEN
            in_latch <= "0000000";
        ELSE
            IF (cnv_clk'EVENT AND cnv_clk = '1') THEN
                in_latch(6) <= cnv_in;
                in_latch(5 downto 0) <= in_latch (6 downto 1);
                -- x = 171 OCT = 1111001 BIN
                x_out <= in_latch(6) XOR in_latch(5) XOR in_latch(4)
                         XOR in_latch(3) XOR in_latch(0);
                -- y = 133 OCT = 1011011 BIN
                y_out <= in_latch(6) XOR in_latch(4) XOR in_latch(3)
                         XOR in_latch(1) XOR in_latch(0);
            END IF;
        END IF;
    END PROCESS;
END rtl;
```

```
-- author: kaohsiung chuah
-- date: 24/03/2003
-- filename: par_ser8.vhd
-- description: this is a 8-bit parallel-serial converter
--              P_Sload is 1 to load codeword

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY par_ser8 IS
    PORT
    (
        P_Sclk      : IN STD_LOGIC;
        P_Sclr      : IN STD_LOGIC;
        P_Sload     : IN STD_LOGIC;
        P_Sin       : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        P_Sout      : OUT STD_LOGIC
    );
END par_ser8;

ARCHITECTURE rtl OF par_ser8 IS
    SIGNAL in_sig    : STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
    PROCESS (P_Sclk, in_sig, P_Sclr)
    BEGIN
        P_Sout <= in_sig(7);
        IF (P_Sclr = '0') THEN
            in_sig <= "00000000";
        ELSE
            IF (P_Sclk'EVENT AND P_Sclk = '1') THEN
                IF (P_Sload = '1') THEN
                    -- load codeword
                    in_sig <= P_Sin;
                ELSE
                    in_sig(7 downto 1) <= in_sig(6 downto 0);
                END IF;
            END IF;
        END IF;
    END PROCESS;
END rtl;
```

Module: puncture

Description: Puncturing


selector
sel_en
sel_Xin
sel_Yin
sel_Iin
sel_Qin
sel_Iout
sel_Qout
inst1
punc_sel0
punc_XYin[0]
punc_XYin[1]
punc_IQout[0]
punc_IQout[1]
punc_IQoutsel[0]
punc_IQoutsel[1]
punc_sel0
punc_bitclk
punc_byteclk
punc_clr
punc_Xin
punc_Yin
INPUT VCC
punc_XYin[0]
punc_XYin[1]
OUTPUT
P_Sout
punc_IQoutsel[0]
punc_IQoutsel[1]
punc_wrreq
punc_rdreq
punc_load
punc_Iout
punc_Qout
par_ser2
P_Sclk
P_Sclr
P_Sload
P_Sin[1..0]
P_Sout
inst2
punc_bitclk
punc_clr
punc_byteclk
punc_XYin[1..0]
P_Sout
ser_par2_ctrl
S_Pbitclk
S_Pclr
S_Pin
S_Pctrl
S_Pout[1..0]
inst5
punc_bitclk
punc_clr
P_Sout
punc_wrreq
punc_IQout[1..0]
punc_ctrl
punc_bitclk
punc_clr
punc_wrreq
punc_rdreq
punc_load
inst
punc_bitclk
punc_clr
punc_wrreq
punc_rdreq
punc_load

```
-- author: kaohsiung chuah
-- date: 21/03/2003
-- filename: par_ser2.vhd
-- description: this is a 2-bit parallel-serial converter
--              P_Sload is 1 to load codeword

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY par_ser2 IS
    PORT
    (
        P_Sclk      : IN STD_LOGIC;
        P_Sclr      : IN STD_LOGIC;
        P_Sload     : IN STD_LOGIC;
        P_Sin       : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        P_Sout      : OUT STD_LOGIC
    );
END par_ser2;

ARCHITECTURE rtl OF par_ser2 IS
    SIGNAL in_sig   : STD_LOGIC_VECTOR(1 DOWNTO 0);
BEGIN
    PROCESS (P_Sclk, in_sig, P_Sclr)
    BEGIN
        P_Sout <= in_sig(0);
        IF (P_Sclr = '0') THEN
            in_sig <= "00";
        ELSE
            IF (P_Sclk'EVENT AND P_Sclk = '1') THEN
                IF (P_Sload = '1') THEN
                    -- load codeword
                    in_sig <= P_Sin;
                ELSE
                    -- shift bits
                    in_sig(0) <= in_sig(1);
                END IF;
            END IF;
        END IF;
    END PROCESS;
END rtl;
```

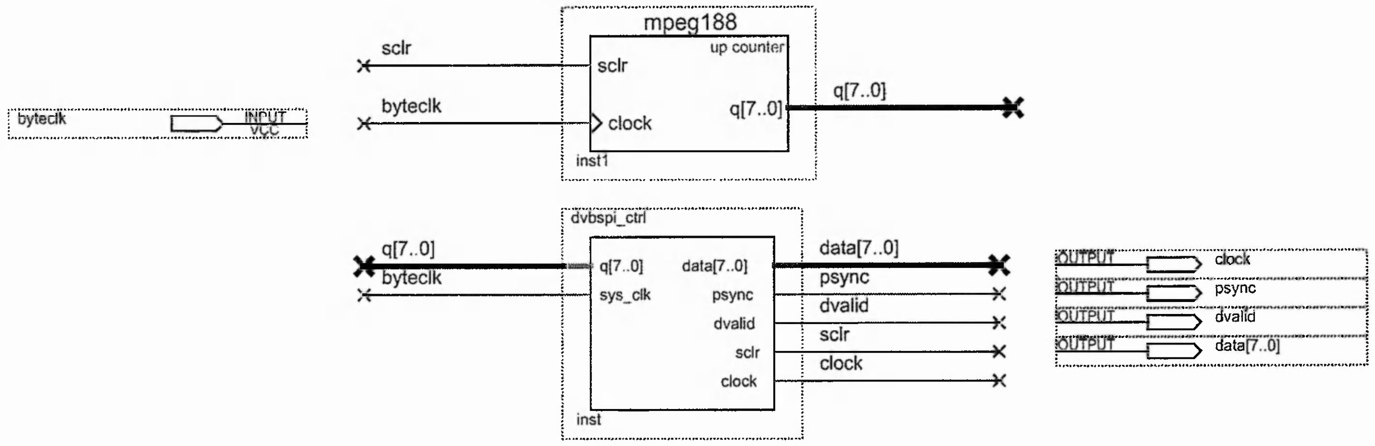
```
-- author: kaohsiung chuah
-- date: 05/02/2004
-- filename: punc_ctrl.vhd
-- description: this controls puncturing process by
--              punc_wrreq sending 0s to puncture bits
--              punc_count to label clock cycles
--              punc_load is not in use

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY punc_ctrl IS
    PORT
    (
        punc_bitclk     : IN STD_LOGIC;
        punc_clr        : IN STD_LOGIC;
        punc_wrreq      : OUT STD_LOGIC;
        punc_load       : OUT STD_LOGIC
    );
END punc_ctrl;

ARCHITECTURE rtl OF punc_ctrl IS

BEGIN
    PROCESS (punc_bitclk, punc_clr)
        -- punc_count to label clock cycles
        VARIABLE punc_count     : INTEGER RANGE 0 TO 3;
    BEGIN
        IF (punc_clr = '0') THEN
            punc_count := 0;
        ELSIF (punc_bitclk'EVENT AND punc_bitclk = '1') THEN
            punc_count := punc_count + 1;
            IF (punc_count = 3) THEN
                punc_count := 0;
                punc_load <= '1';
                punc_wrreq <= '0';
            ELSE
                punc_load <= '0';
                punc_wrreq <='1';
            END IF;
        END IF;
    END PROCESS;
END rtl;
```

```
-- author: kaohsiung chuah
-- date: 05/02/2004
-- filename: ser_par2_ctrl.vhd
-- description: this serial-parallel converter punctures when
--              S_Pctrl is 0, as it stops shifting input bits
--              S_Pout load its values as I and Q
--              bits that were not shifted into the converter is pun
ctured

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ser_par2_ctrl IS
    PORT
    (
        S_Pbitclk   : IN STD_LOGIC;
        S_Pclr      : IN STD_LOGIC;
        S_Pin       : IN STD_LOGIC;
        S_Pctrl     : IN STD_LOGIC;
        S_Pout      : OUT STD_LOGIC_VECTOR(1 DOWNTO 0)
    );
END ser_par2_ctrl;

ARCHITECTURE rtl OF ser_par2_ctrl IS
    SIGNAL in_sig       : STD_LOGIC;
    SIGNAL outsig       : STD_LOGIC_VECTOR(1 DOWNTO 0);
BEGIN
    PROCESS (S_Pbitclk, S_Pctrl)
    BEGIN
        -- S_Pout <= outsig;
        IF (S_Pclr = '0') THEN
            outsig <= "00";
            in_sig <= '0';
        ELSIF (S_Pctrl = '1') THEN
            IF (S_Pbitclk'EVENT AND S_Pbitclk = '1') THEN
                in_sig <= S_Pin;
                -- shifting input bits
                outsig(0) <= in_sig;
                outsig(1) <= S_Pin;
            END IF;
        ELSIF (S_Pctrl = '0') THEN
            IF (S_Pbitclk'EVENT AND S_Pbitclk = '1') THEN
                -- S_Pout load its values as I and Q
                S_Pout <= outsig;
            END IF;
        END IF;
    END PROCESS;
END rtl;
```

```
-- author: kaohsiung chuah
-- date: 25/06/2004
-- filename: selector.vhd
-- description: this enables bypass of puncturing process
--              to switch between 1/2 and 3/4 rate
--              when sel_en is 0, X and Y is connected to I and Q
--              when sel_en is 1, I and Q is punctured output

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY selector IS
    PORT
    (
        sel_en          : IN STD_LOGIC;
        sel_Xin         : IN STD_LOGIC;
        sel_Yin         : IN STD_LOGIC;
        sel_Iin         : IN STD_LOGIC;
        sel_Qin         : IN STD_LOGIC;
        sel_Iout        : OUT STD_LOGIC;
        sel_Qout        : OUT STD_LOGIC
    );
END selector;

ARCHITECTURE synth OF selector IS
BEGIN
    PROCESS (sel_en)
    BEGIN
        IF (sel_en = '0') THEN
            -- select X and Y input
            sel_Iout <= sel_Xin;
            sel_Qout <= sel_Yin;
        ELSE
            -- select punctured I and Q
            sel_Iout <= sel_Iin;
            sel_Qout <= sel_Qin;
        END IF;
    END PROCESS;
END synth;
```

Module: dvbspisource_v2

Description: Generates DVB SPI MPEG source for testing


byteclk
INPUT
VCC
sclr
byteclk
mpeg188
up counter
sclr
clock
q[7..0]
inst1
q[7..0]
dvbspi_ctrl
q[7..0]
byteclk
q[7..0]
sys_clk
data[7..0]
psync
dvalid
sclr
clock
inst
data[7..0]
psync
dvalid
sclr
clock
OUTPUT
clock
OUTPUT
psync
OUTPUT
dvalid
OUTPUT
data[7..0]

```
-- author: kaohsiung chuah
-- date: 19/03/2004
-- filename: dvbspi_ctrl.vhd
-- description: this generates 188-byte mpeg packets with
--              the last 16 null-bytes

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY dvbspi_ctrl IS
    PORT
    (
        q           : IN STD_LOGIC_VECTOR(7 downto 0);
        sys_clk     : IN STD_LOGIC;
        data        : OUT STD_LOGIC_VECTOR(7 downto 0);
        psync       : OUT STD_LOGIC;
        dvalid      : OUT STD_LOGIC;
        sclr        : OUT STD_LOGIC;
        clock       : INOUT STD_LOGIC
    );
END dvbspi_ctrl;

ARCHITECTURE dvbspi_ctrl_architecture OF dvbspi_ctrl IS
    CONSTANT SYNC        : STD_LOGIC_VECTOR := "01000110";
    CONSTANT BYTE187     : STD_LOGIC_VECTOR := "10111010";
    CONSTANT BYTE203     : STD_LOGIC_VECTOR := "11001010";
    SIGNAL out_sig       : STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL psync_sig     : STD_LOGIC;
    SIGNAL zero_out      : STD_LOGIC;

BEGIN
    clock <= sys_clk;
    out_sig <= q;

    -- detect clock at 71 and 187, to output psync and reset counter
    PROCESS (sys_clk)
    BEGIN
        IF (sys_clk'EVENT AND sys_clk = '1') THEN
            IF (q = SYNC) THEN
                psync_sig <= '1';
                sclr <= '0';
                --out_sig <= "01000111";
            ELSIF (q = BYTE187) THEN
                psync_sig <= '0';
                sclr <= '1';
            ELSIF (q = BYTE203) THEN
                sclr <= '0';
                psync_sig <= '0';
            ELSE
                psync_sig <= '0';
                sclr <= '0';
            END IF;

            -- null-bytes flag controls
            IF (q = "00110110") THEN
                zero_out <= '1';
            ELSIF (q = "01000110") THEN
                zero_out <= '0';
            ELSE
                zero_out <= zero_out;
            END IF;
        END IF;
        IF (sys_clk'EVENT AND sys_clk = '0') THEN
            -- un-comment this section to enable null-bytes
            -- data <= "01000111";
            --IF ( zero_out = '1') THEN
            --  data <= "00000000";
            --  dvalid <= '0';
            --ELSE
                data <= out_sig;
                psync <= psync_sig;
                dvalid <= '1';
            --END IF;
        END IF;
    END PROCESS;
END dvbspi_ctrl_architecture;
```