Jake Mercer

MEng (Hons) Final Year Project Report:

Verilog Open-Source Ethernet MAC

May 2017

i. Abstract

This report sets out the process for the design and implementation of an Ethernet MAC conforming to IEEE 802.3 as part of a larger project to produce a network device testing unit. The resulting MAC will be made open source, allowing for anyone to use it for free in their designs.

The MAC is designed with simplicity and low resource usage in mind and is compared to existing implementations, with the MAC in this report coming out on top in terms of usage, however with a reduced feature set.

The project was initially an exercise in understanding Ethernet and the MAC specification within the standard, as well as understanding implementation of logic on FPGAs. This was followed by an initial design of the MAC and coding in Verilog for receive and transmit functionality. Additional functionality, such as station management and media independent interfaces were added later.

Testing was completed first by simulation, with automated test benches now being included in the repository. After the initial implementation was made, testing was done using the integrated logic analyser in Vivado, followed by physical testing of the design with another network testing device.

Future plans are laid out for using this MAC as part of the larger network device tester and work on the project continues.

ii. Specification

The objectives of this project are:

- Selection of an appropriate FPGA development board/kit to develop the traffic testing module.
- Familiarisation with existing MAC cores e.g. Tri-Mode Ethernet MAC Core, TEMAC by Xilinx, etc.
- Familiarisation with the board hardware:
- Implementing an Ethernet MAC IP core on the development board to include:
- FPGA Design of a fully parameterised Ethernet MAC.
- Flexible design allowing for parameterised bus widths, clock frequencies, etc.
- Fully conforming to IEEE 802.3.
- Usable at any of the line rates available for Ethernet.
- FPGA Design of a fully parameterised reconciliation sublayer to provide MII, RMII, GMII and RGMII interfaces.

iii. Acknowledgements

Foremost, I would like to acknowledge Dr John McAllister, who supervised the project and without which the MAC would have never been fully implemented. My thanks go to Dr Kieran McLaughlin as well for moderating the project and providing a fresh pair of eyes on it.

Thanks must also go to John Kennedy and Eddie Millsopp of Asidua, who were the instigators of the project, as well as the rest of the Asidua team, who provided help and advice throughout the design step.

iv. List of Figures

v. Declaration of Originality

I declare that this report is my original work except where stated.

………………………………………………………………….

Contents

1 Introduction

Civica Digital Solutions Ireland's (CDSI)/Asidua's Device Services (DS) department design, write and test embedded software for bridge routers; testing of such software requires the ability to power-cycle the Device Under Test (DUT), have both direct serial and network control of a shell on the DUT, and the ability to send specific packets at specific rates to the network interfaces of the DUT.

Currently, DS relies on IP Power Switches to power cycle DUTs over the network; whilst these are convenient, they have many drawbacks, namely the relatively short life of on-board relays and the internal power supply. This is due in no small part, to the power switches being operated above and beyond their rated capabilities. Modifications have been made to the IP Power Switches, however, it is not practicable to continue to fix the symptoms of bad design and usage. When a relay, or power supply dies, it usually requires a significant investment of time to figure out that the issue is with the IP Power Switch and not with the DUT itself, and this translates to less development time.

The DS lab has no terminal servers, so the management connection to the DUT is solely over Telnet, this has drawbacks in that, to see the U-Boot boot process happening (before telnetd is loaded) one has to be present in the lab with a serial connection to a laptop plugged in.

Network testing of the DUTs is currently performed by the Spirent Test Centre (STC) and represents a single point of failure for the lab. If the STC is unavailable, very few automated tests can be run. The number of ports on the STC are limited also, and there are more DUTs than can be tested with the STC. The STC represents a significant expense due to licensing and hardware, such as cards and Small Form-Factor Pluggable (SFP) transceivers. The STC operates at Gigabit line rate on each port and can generate and analyse traffic at that line rate with few limitations. Spirent also offer several expensive automated test packages for protocols such as Ethernet Operations, Administration and Maintenance (EOAM), Bidirectional Forwarding Detection (BFD), and Spanning Tree Protocol (STP) to name a few.

In addition, if a DUT is off-site e.g. at a customer's location, it is unusable for testing as it is disconnected from the lab, and it is not feasible to send test traffic over the internet or fully integrate with customer equipment.

1.1 Aims & Objectives

It is proposed that a portable & remote network device testing unit with an RS232 serial server, a power switch server, and an FPGA (for network traffic generation and analysis) for bridge router development be designed and manufactured to streamline the testing & development process. The testing device will be able to carry out the aforementioned tasks (power-cycling, serial access over a network, network traffic testing). This unit will be small and inexpensive, and could be sent along with a DUT to a customer to allow for remote testing.

An FPGA development board will be selected to facilitate development of the traffic testing module. An initial design overview of the traffic testing module is shown in Figure 1.

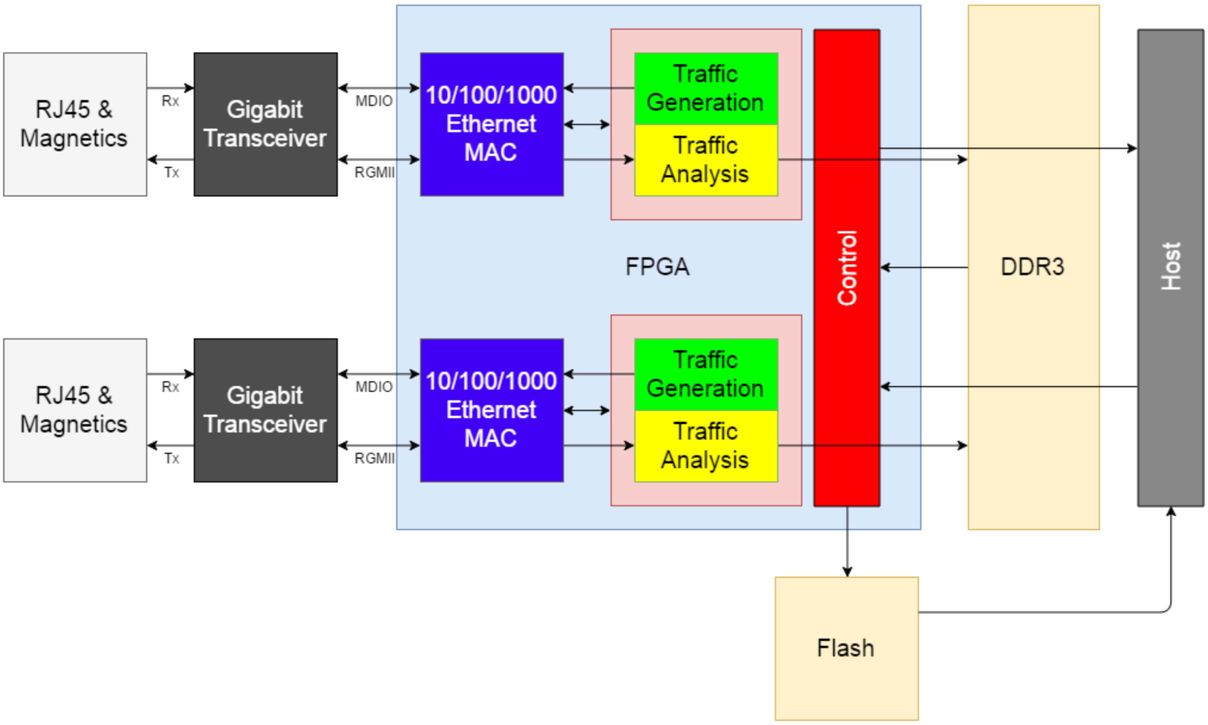


Figure 1 – Design overview of proposed Ethernet tester.

Part of the project will deal with the design of the FPGA logic for transmitting a packet, starting off with the basics of sending and receiving a packet, and ensuring packets are not malformed. The complexity of the logic will be gradually increased to include TX and RX FIFOs, and eventually being able to send an assortment of packets correctly, calculating checksums where necessary, the end goal being a fully functional Ethernet Media Access Controller (MAC), which is the topic of this report.

1.2 Scope

Due to time constraints the scope of this report will be limited to:

- The design and implementation of an Ethernet MAC IP fully conforming to IEEE 802.3 MAC specification.
- The design and implementation of PHY control IP using Management Data Input/Output (MDIO).
- The design and implementation of a Reconciliation Sublayer (RS) conforming to the appropriate IEEE 802.3 RS specification.

1.3 Motivation

There is something to be said about the motivation behind designing and implementing an Ethernet MAC, as there are a few implementations available and the old adage of "don't reinvent the wheel" could be considered applicable. The alternative implementations are mentioned on page 57 of this report; these designs are proven and in working systems today which means that they could be dropped into a design and the designer could be fairly confident in their operation. On the other hand, these implementations aren't designed with the end goal of being used in a network testing device and tend to bundle functions that are not necessarily part of the IEEE 802.3 standard and may not exactly represent the tests which we wish to carry out. Instead these designs are intended to be fully functional Ethernet front-ends which simply add network functionality as an accessory to a product, as opposed to being the main component in that product. A network testing device calls for an Ethernet MAC which rigorously conforms to IEEE 802.3 and does no more and no less. This design is intended to be an extremely pure Ethernet MAC.

In the future, functions that come bundled with other implementations such as IEEE 802.2 flow control, and associated Link Layer Control (LLC) utilities would be designed as a module of their own and could be specifically instantiated as part of a test case. This helps to keep the components of the test modular and efficient. As you'll see in the section below, Ethernet has its roots in being open, and it is in this spirit that this MAC will be open-source and free for anyone to use.

2 Background Information

2.1 A Brief History of Ethernet

Ethernet is ubiquitous in the telecommunications world, everything from the internet to your car uses it to transfer messages from one device to another [1]. Without Ethernet, the planet would be a lot less connected and communications wouldn't have been able to make the leaps in technology that we can see today. It is the precursor to many other communications systems such as PCIe and RapidIO [2], which are similar in nature, but are designed for different tasks; it's one of the most widely adopted technologies in all manner of products; and it's one of the most rapidly progressing pieces of engineering available, with speeds reaching 400 Gbit/s in IEEE research clusters, and soon to become part of the Ethernet standard [3].

1973 saw the birth of Ethernet. Inspired by the earlier Aloha network, in which Hawaiian Islands were connected via radio, Bob Metcalfe of Xerox recognised the pitfalls and developed a new system. Aloha used a simple setup where any radio could transmit at any time, and an acknowledgement was sent on reception of a message. If the transmitting station did not receive an acknowledgement, it was assumed that multiple stations had attempted transmission and a collision had occurred. Upon detection of a collision, transmitting stations backed off for a random amount of time before retrying. With increased traffic came an increased number of collisions, resulting in a maximum channel utilisation of around 18% [4].

The new system that Metcalfe designed became Ethernet. It improved upon Aloha by having each station listen to the channel before trying to transmit, this massively reduced the number of collisions and made it easier for multiple stations to share a channel, this formed the Carrier Sense Multiple Access (CSMA) protocol, which was further improved by adding Collision Detection (CD), as in Aloha albeit implemented differently. You will often see this mentioned as CSMA/CD. The backoff algorithm was also improved upon, putting Ethernet well ahead of any networking technology of the time [4].

Metcalfe saw the importance of interconnectivity of devices and was a forward-thinking individual. He convinced Xerox to release Ethernet to the world as an open standard, allowing anyone to copy and use it, and formed a consortium of vendors to manage the standard, eventually being adopted by the IEEE and recognised as an ISO standard [4].

In terms of the physical technology, Ethernet was originally run over coaxial cabling, with a bus topology. Stations connected to the network via taps into the coax and shared the medium. This came with issues: if the backbone of the network was damaged, connectivity was lost and finding the fault was difficult. With the advent of twisted pair, it became possible for the network topology to change from bus to star, making for a much less painful installation process. Most Ethernet installations today still use the twisted pair technology (see Figure 2) and advancements have been made in using fibre optics to carry Ethernet as well [4].

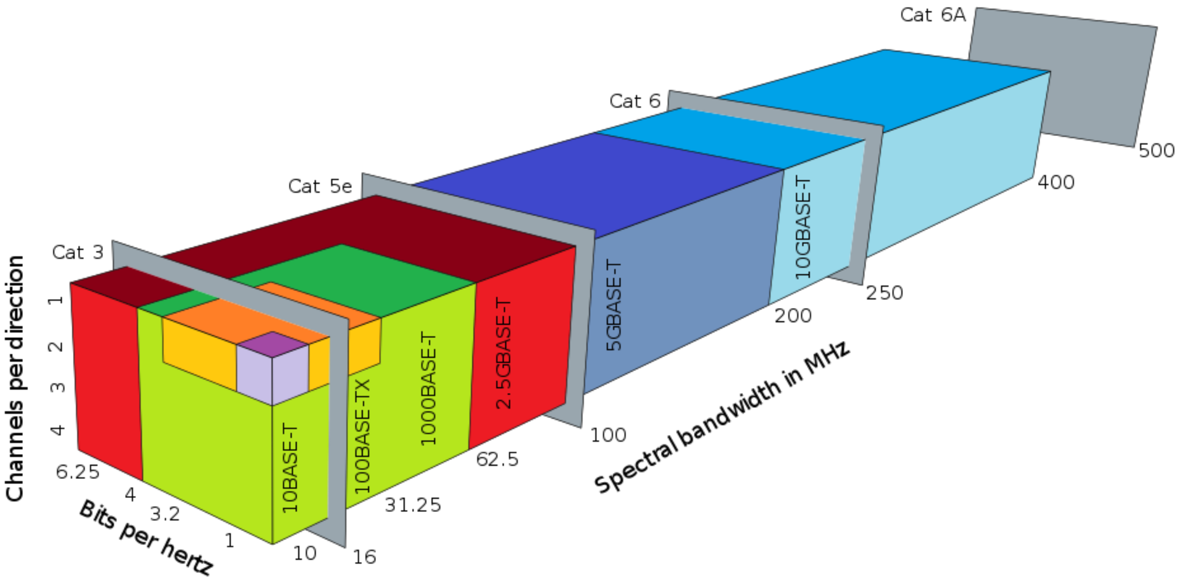


Figure 2 – Cat 3 – Cat 6A Unshielded Twisted Pair (UTP) cabling and the Ethernet standards they relate to. [5]

The 1980 standard of Ethernet provided a means of communicating at 10Mb/s and the 90s saw Fast Ethernet (FE) come alive with speeds of up to 100Mb/s. With FE came auto-negotiation, allowing for backwards compatibility with the slower 10Mb/s devices. Most Network Interface Cards (NIC) these days operate at 1 Gb/s in full duplex, whilst still being able to operate at the slower speeds in both full and half duplex modes. In high-end Ethernet installations 10 Gb/s is not uncommon and 40 and 100 Gb/s network switches are now available for purchase. At the time of writing, a 200 and 400 Gb/s research cluster is putting the final touches on the Terabit Ethernet (TbE) standard for release at the end of the year [6].

Note: 2.5 GbE and higher operate in full duplex mode only, the CSMA/CD protocol no longer applies.

Ethernet is not the only major networking technology, but it is certainly the most popular with opportunities arising for the DSL lines of old to be replaced by Ethernet in the First Mile (EFM) and provider networks increasingly using Ethernet for applications such as mobile backhaul. However, Ethernet may not have reached the critical acclaim of other standards/Request for Comments (RFC), such as RFC 1149 – IP over Avian Carriers [7], or the more recent RFC 7511 – Scenic Routing for IPv6 [8].[1]

2.2 IEEE 802.3 – The Ethernet Standard

The Ethernet standard has expanded and grown over the years, but its basic principles have remained the same. IEEE 802.3 is split into 6 sections at the time of writing, each section corresponding to a speed increase i.e. section 1 covers the MAC sublayer for all modes of operation and 10 Mb/s operation; section 2 goes over 100 Mb/s operation; section 3, 1000 Mb/s; and so on. Within each section, there are clauses dedicated to the interface to the MAC sublayer, as well as each of the possible underlying physical configurations. The standard is intentionally designed to be analogous to layers 1 and 2 of the Open Systems Interconnection (OSI) model [9].

2.2.1 Ethernet Packet Structure

Sending a data payload over Ethernet requires conformance to the Ethernet packet structure, as defined in IEEE 802.3. The payload is encapsulated by the Ethernet packet and, while there is some overhead involved in including the Ethernet header, it provides vital source and destination information, as well as error detection [9]. The structure is shown in Figure 3.

[1] I very much encourage you to go read these RFCs as they prove that some engineers do, in fact, have a sense of humour, and probably too much time on their hands.

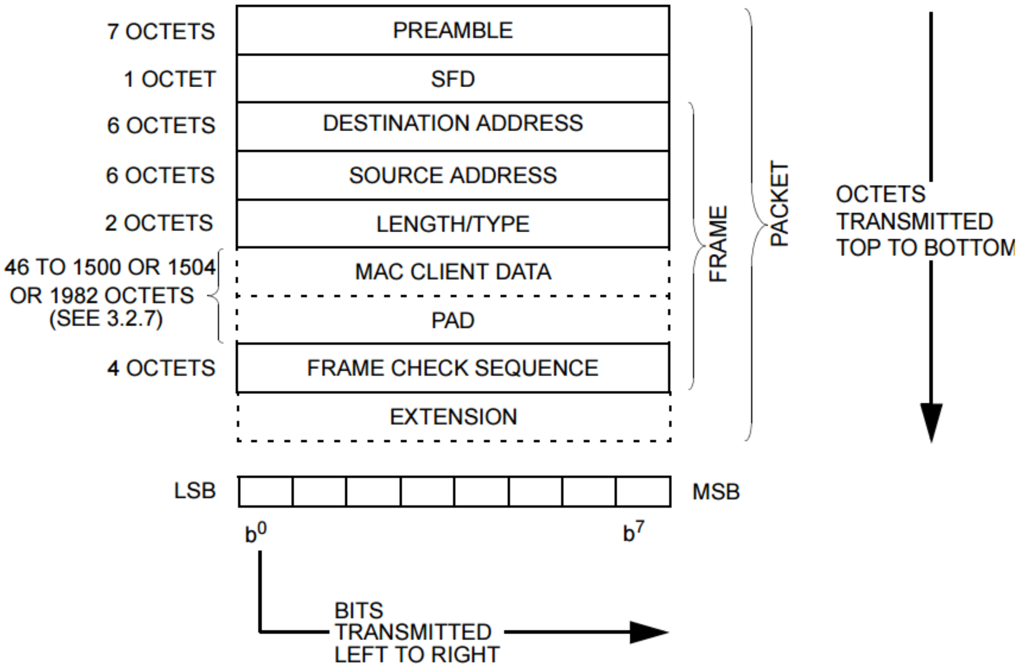


Figure 3 - Ethernet frame & packet format [9]

You may often hear the terms Ethernet packet and Ethernet frame used interchangeably but the distinction between the two has to be made. The Ethernet packet is all of the fields seen in the figure, whilst an Ethernet frame is a subset of this, defined as the data after the start of frame delimiter but before the frame check sequence. The reason for this is that the MAC client (upper layers) should never see the preamble and start of frame delimiter, and only in rare circumstances will see the frame check sequence.

2.2.2 Media Access Control Sublayer

In order to perform the CSMA/CD operations and send a packet, the concept of the Media Access Controller (MAC) sublayer was created (Figure 4). The job of the MAC is to listen to the media, acquire the media, transmit or receive Ethernet packets, and detect errors, along with a few other functions, such as recording the number of packets it has received. The MAC as it sits relevant to the OSI model is made apparent in Figure 4.

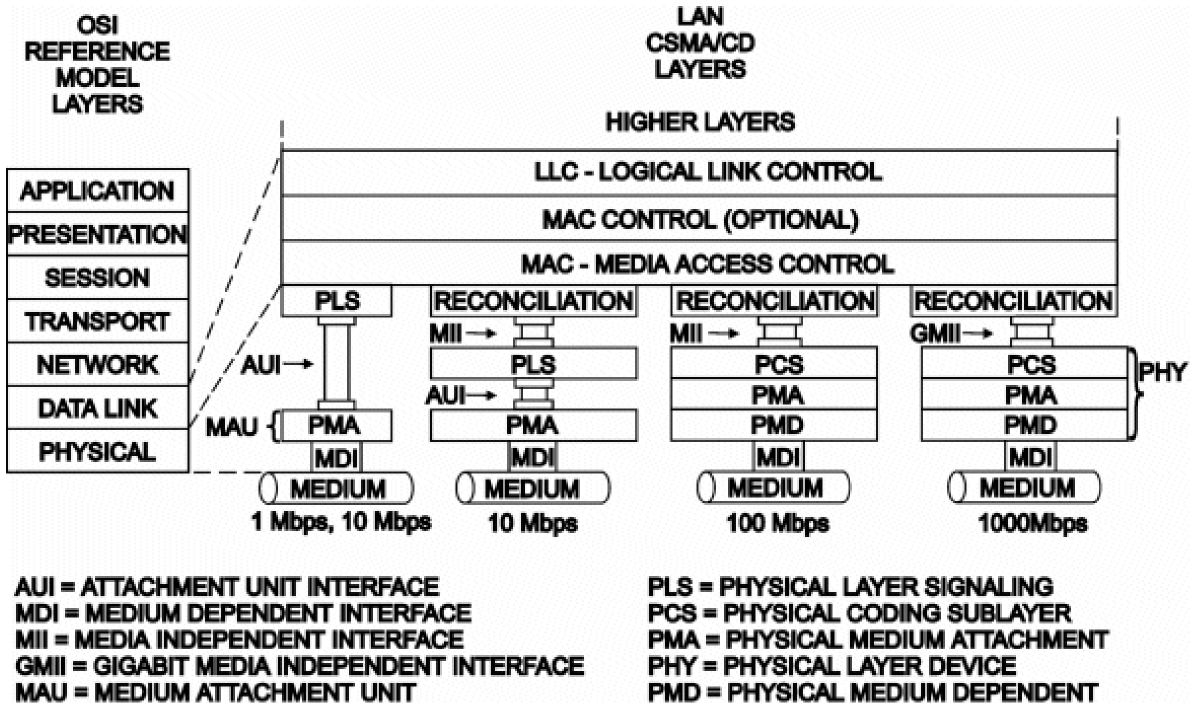

OSI REFERENCE MODEL LAYERS
LAN CSMA/CD LAYERS
HIGHER LAYERS
APPLICATION
PRESENTATION
SESSION
TRANSPORT
NETWORK
DATA LINK
PHYSICAL
LLC - LOGICAL LINK CONTROL
MAC CONTROL (OPTIONAL)
MAC - MEDIA ACCESS CONTROL
PLS
RECONCILIATION
RECONCILIATION
RECONCILIATION
MII
MII
GMII
AUI
PLS
PCS
PCS
PHY
AUI
PMA
PMA
MAU
PMA
PMA
PMD
PMD
MDI
MDI
MDI
MDI
MEDIUM
MEDIUM
MEDIUM
MEDIUM
1 Mbps, 10 Mbps
10 Mbps
100 Mbps
1000Mbps
AUI = ATTACHMENT UNIT INTERFACE
MDI = MEDIUM DEPENDENT INTERFACE
MII = MEDIA INDEPENDENT INTERFACE
GMII = GIGABIT MEDIA INDEPENDENT INTERFACE
MAU = MEDIUM ATTACHMENT UNIT
PLS = PHYSICAL LAYER SIGNALING
PCS = PHYSICAL CODING SUBLAYER
PMA = PHYSICAL MEDIUM ATTACHMENT
PHY = PHYSICAL LAYER DEVICE
PMD = PHYSICAL MEDIUM DEPENDENT

Figure 4 – Ethernet layers as they relate to the OSI Model [9]

2.2.2.1 Basic Operating Principles

2.2.2.1.1 Preamble & Start of Frame Delimiter

The preamble is a 7-octet field consisting of alternating bits:

10101010 10101010 10101010 10101010 10101010 10101010 10101010

It is used to synchronise the Physical Layer (PHY) circuitry with the packet timing; at the MAC layer this signal, along with a data valid signal denotes an incoming packet. The Start of Frame Delimiter (SFD) follows this and has the sequence:

10101011

This indicates the start of the frame data. A transmitting station's MAC prepends the preamble and SFD to the frame.

2.2.2.1.2 MAC Address

There are two MAC address fields in the Ethernet frame, destination and source. The destination address is the address of the desired receiving station and the source address is the address of the transmitting station. A MAC address is a 48-bit field and certain configurations represent different meanings for the network. The first bit of the address indicates whether the address is an individual station or a group of stations, an address for an individual station is often called unicast, while group addresses are known as multicast or broadcast. Multicast addresses can be sent to a subset of the stations on a network, depending on the circumstances, while the broadcast address is intended to be received by all stations. The second bit of the MAC address indicates whether the address is globally or locally administered. MAC addresses are pre-designated for network devices and manufacturers can apply to the IEEE for an Organisationally Unique Identifier (OUI) for building a MAC address. The format for a MAC address can be seen in Figure 5.

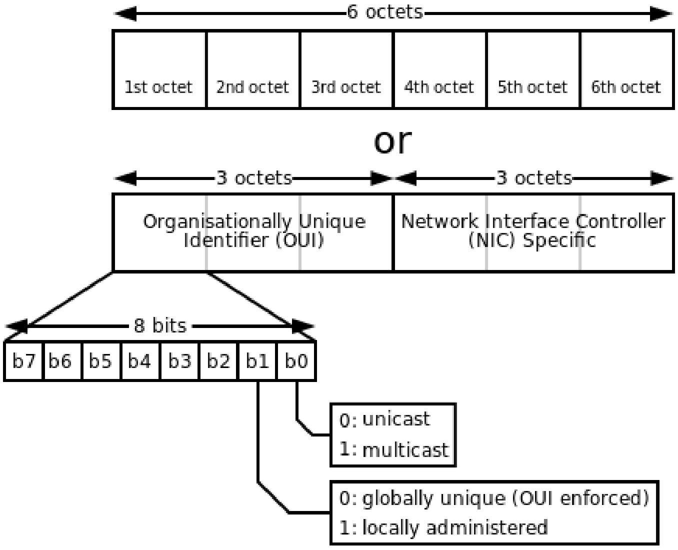


Figure 5 - MAC address [10]

Note: Broadcast addresses are considered locally administered and are represented by an address of all ones. Source addresses can never be group addresses.

2.2.2.1.3 Length/Type

The length/type field provides information on the MAC client data which follows it and, depending on its value, takes one of two forms: length or type.

If the value is less than the maximum allowable MAC Service Data Unit (SDU) size (1500 octets nominal) the field is interpreted as a length field which is then used to determine if the frame is valid or not: if the length does not match the SDU length, then the frame is discarded.

If the value is greater than 1536 then the field is interpreted as a type field, and indicates the type of SDU.

Some examples of available types are noted in Table 1. You will often see this field written down as the EtherType field.

Table 1

Value (as seen at the MAC layer)	Protocol
0x0800	IPv4
0x86DD	IPv6
0x8847	MPLS
0x8902	EOAM

2.2.2.1.4 Sizing

If a received packet's size doesn't fall within certain limits, then the packet is discarded by the MAC. The preamble and SFD are not considered in the length calculation. The basic limits are as follows:

- Maximum – 1518 octets.
    - $Max\ SDU\ Length + 2(Address\ Length) + EtherType\ Length + FCS\ Length$
- Minimum – 64 octets.

There are certain types of Ethernet frame that are allowed to exceed the basic 1500 octet limit of SDU size. These are Q-tagged frames and envelope frames [9].

Q-tagged frames are frames with the optional VLAN tag inserted between the source address and the type/length field. The VLAN tag is 4 octets wide and so the maximum SDU size increases from 1500 to 1504 octets.

Envelope frames are frames with extra headers, such as MPLS, used in higher layer encapsulation protocols, and encapsulate an Ethernet frame. The encapsulation protocol overhead may add up to an extra 482 octets allowing for a maximum SDU size of 1982 octets. The encapsulated frame's SDU size limit remains at 1500 octets.

The SDU size may also be referred to as the Maximum Transmission Unit (MTU) [9].

2.2.2.1.5 Padding

If the frame that is to be sent is too short (less than 64 octets) then the MAC pads the SDU with data to increase it to the minimum size. The content of the padding is unspecified, however I know of no implementations that transmit anything other than zeroes as padding.

The padding length (as specified in IEEE 802.3) can be calculated as follows [9]:

$$\max[0, minFrameSize - (clientDataSize + 2 * addressSize + 48)]\ bits$$

Equation 1

2.2.2.1.6 Frame Check Sequence

The Frame Check Sequence (FCS) provides error detection in the packet. It consists of a 32 bit Cyclic Redundancy Check (CRC) and is calculated as a function of the contents of the Ethernet frame i.e. excluding the preamble, SFD and FCS fields. If interference or noise occurs during transmission and flips a bit, then the calculation at the receiver will fail and the packet will be discarded.

The generating polynomial for Ethernet is [9]:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$$

Equation 2

There are a few points to be made regarding bit and byte order for Ethernet CRC as the CRC is transmitted most significant bit first, and in reverse endianness to the rest of the packet.

2.2.2.1.7 Inter-Frame/Inter-Packet Gap

A gap between transmissions is inserted called the Inter Packet Gap (IPG) or Inter Frame Gap (IFG) (used interchangeably). The IFG is present to allow recovery time for devices connected to the network. The IFG has a length of 96 bits for all Ethernet implementations and can be extended at the expense of throughput. Due to network delays, and clock issues it is possible for the IFG to shrink, the amount of shrinkage is dependent on the data rate and this must be taken into account when designing an Ethernet MAC [9].

2.2.2.2 Half Duplex Operation

2.2.2.2.1 Carrier Sense Multiple Access with Collision Detection

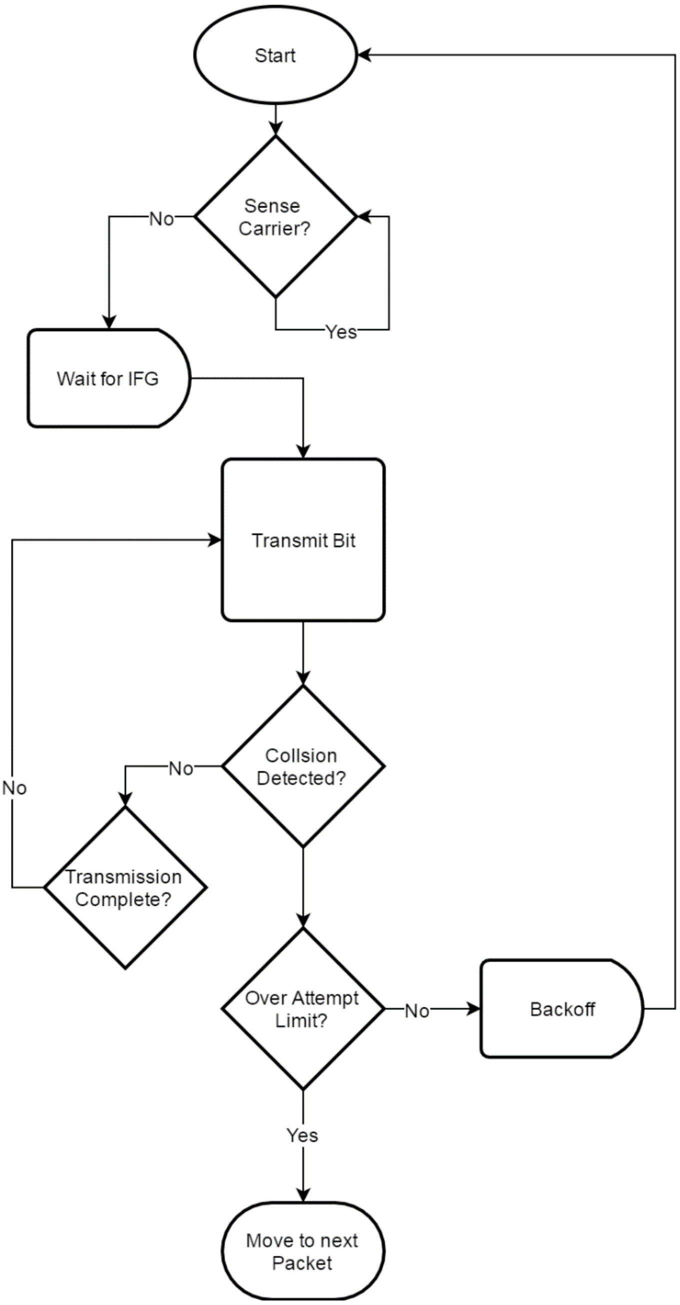


Figure 6 - CSMA/CD flow diagram

In half duplex, the MAC listens for transmissions using the Carrier Sense (CS) signal from the PHY. If CS is asserted, the MAC defers any scheduled transmissions until the CS signal is de-asserted and then waits for the IPG to pass before attempting transmission. A Collision Detect (COL) signal from the PHY alerts the MAC to a collision during the transmission. The MAC then sends a jam signal and performs a backoff operation. The flow diagram in Figure 6 shows the process minus the jam signalling and smaller details.

2.2.2.2.2 Packet Bursting

At 1000 Mb/s operation, and in half duplex mode, packet bursting is allowed. This is when the transmitting station does not relinquish control of the medium after transmission of a packet, instead opting to continue transmitting until a predetermined burst limit is reached. In order to achieve this, the MAC must not allow an idle state to be reached between packets or other stations will sense a free medium and the next packet will not be considered part of the burst and may not be received properly by some stations. The first packet of a burst may be extended, as described below, in order to avoid collisions on subsequent packets of a burst (Figure 7).

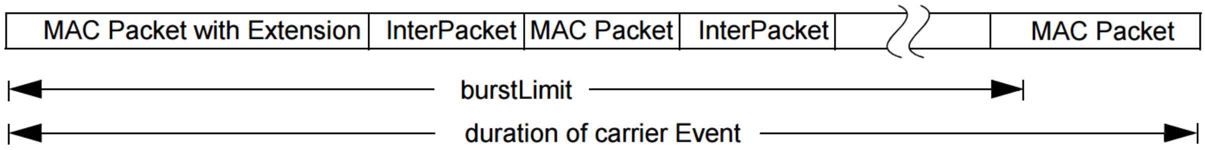


Figure 7 - Packet bursting [9]

2.2.2.2.3 Slot Time

The slot time is the maximum time it can take for a transmitting station to successfully acquire the medium. As such, packet collisions can only occur within a slot time of the start of transmission; by that point the packet will have reached all stations on the network and the PHY circuitry will have asserted CS, meaning that all other stations will defer to the transmitting station. If a COL signal is asserted after a transmission has completed, this is known as a late collision and is usually due to incorrect setup, Ethernet cable lengths exceeded, or a malfunctioning network device. Packets that experience a late collision are not retransmitted.

When the speed of operation increases from 100 Mb/s to 1000 Mb/s the slot time of 512 bit times is no longer sufficient for reasonably sized networks to operate, and as such the slot time is increased to 4096 bit times [9].

2.2.2.2.4 Carrier Extension

At 1000 Mb/s the slot time no longer matches the minimum frame size of 512 bits (64 octets). In order to increase transmission duration to the slot time of 4096 bit times without altering the packet (and therefore maintaining backwards compatibility for other, slower network segments and less capable MACs) a function called carrier extension (Figure 8) can be optionally implemented. This requires the transmission of non-data symbols after a packet to increase the transmissions length to reach the slot time and is only possible if the physical layer below the MAC layer supports transmission of extension bits. If carrier extension is not implemented then there is an increased risk of late collisions occurring for packet sizes below the slot time in length [9].

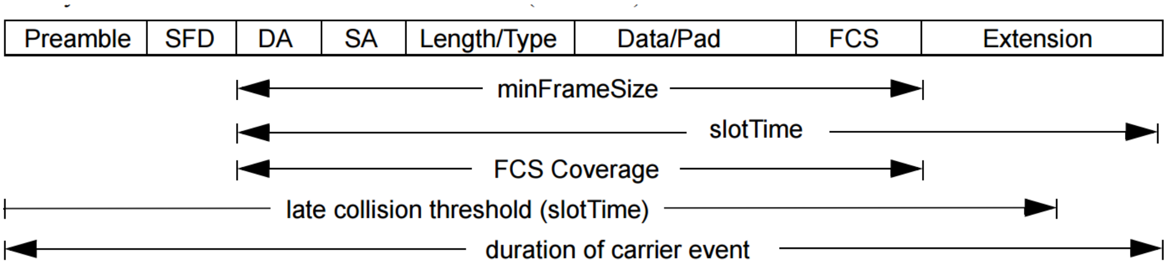


Figure 8 - Carrier extension [9]

2.2.2.2.5 Jamming

If the COL signal is asserted during the sending of a packet, the MAC does not immediately cease transmission. It instead transmits a jamming signal to enforce the collision, making the collision last long enough so that all transmitting stations are made aware of the collision. The jamming signal is 32 bits in length in all cases. Care must be taken not to accidentally send the CRC for the already transmitted fragment in case the receiving MACs interpret this as a valid frame [9].

2.2.2.2.6 Collision Backoff

After the jam signal has completed, transmission stops and the MAC queues the packet for retransmission after a random integer number of slot times have passed. If there is a subsequent collision, transmission is retried up to 10 reattempts at which point no further transmission of the packet is attempted and the MAC transmits the next available packet instead. The delay is calculated as follows [9]:

$$Delay = r(Slot\ Time)$$ *Equation 3*

Where

$$0 \leq r \leq 2^k$$ *Equation 4*

Where

$$k = \min[n, 10]$$ *Equation 5*

2.2.2.3 Full Duplex Operation

Full duplex operation is much less complex than half duplex operation: CSMA/CD no longer applies and features such as packet bursting and carrier extension are not available. Because there is no contention for the media, stations can transmit without regard for the CS or COL signals. The Ethernet packet however, does not change. The IPG is still enforced.

2.2.3 Reconciliation Sublayer

The Reconciliation Sublayer (RS) sits directly below the MAC sublayer (Figure 4) and provides an interface to the PHY as detailed below. The Media Independent Interface (MII) was originally a workstation interface and connected via a cable to the PHY, nowadays the MII is used solely on PCBs and occasionally in internal on-chip designs. The MII, as the name would suggest, is media independent and is intended to allow the MAC sublayer to connect to any physical layer, whether it's electrical (coaxial, twisted pair) or optical (single-mode and multi-mode fibre optic).

2.2.3.1 Media Independent Interface

The earliest of the MIIs, the media independent interface is capable of up to 100 Mb/s operation [9]. Its signals are shown in Figure 9.

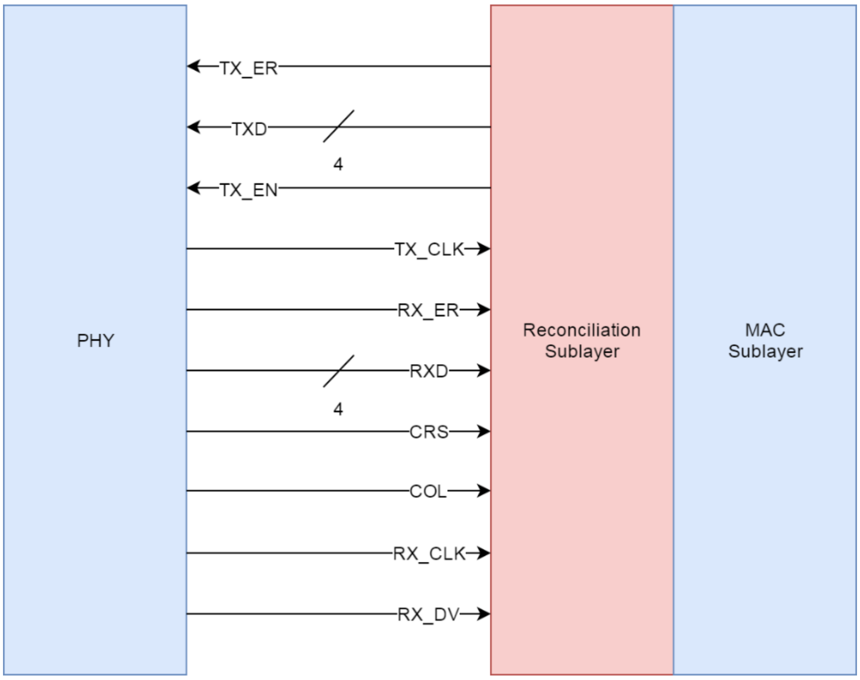


Figure 9 - MII signals

The transmit clock is generated by the PHY and can be either 25 MHz for 100 Mb/s operation, or 2.5 MHz for 10 Mb/s operation. Data is synchronous to its respective clock and transitions on the rising edge of that clock.

When transmitting, the MAC asserts the transmit enable signal and if the MAC wishes to abort the current transmission, or another error occurs, transmit error is asserted. When not transmitting, these signals can be used for other purposes to control the PHY. All of the following figures are in half-duplex mode.

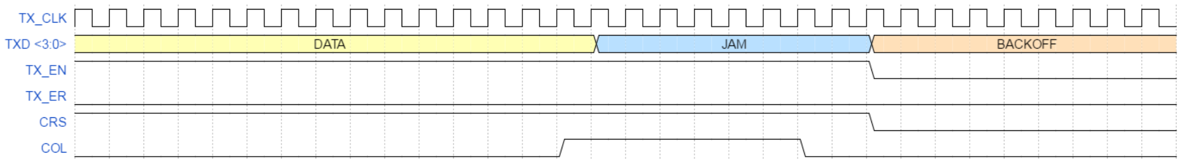


Figure 10 - MII –Transmit – Collision

Figure 10 shows the MII signals during a collision condition. When the MAC detects the COL signal a 32-bit jam signal is sent.

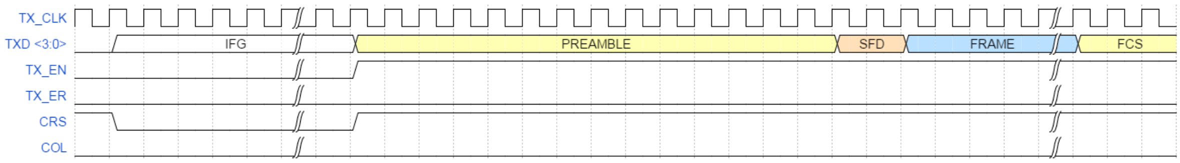


Figure 11 – MII - Transmit – Deferring

Figure 11 shows the MII signals while the MAC defers for the IFG after CS is de-asserted.

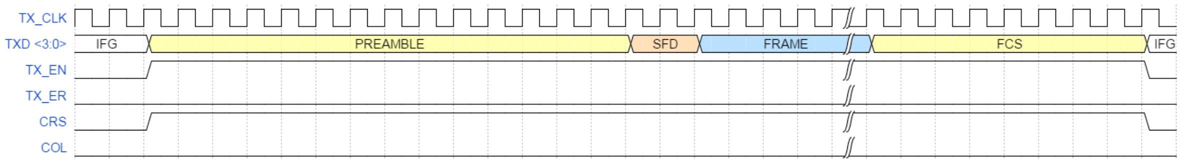


Figure 12 - MII - Transmit - Normal Operation

Figure 12 shows the MII signals during a normal transmit operation.

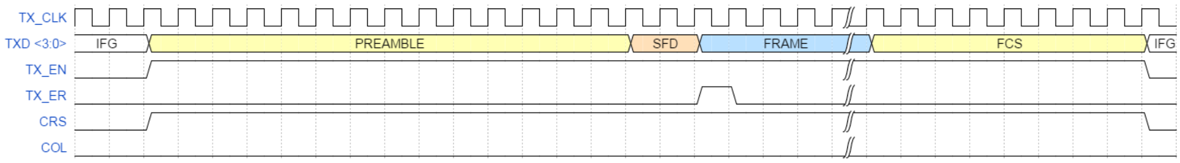


Figure 13 - MII - Transmit - Transmit Error

Figure 13 shows the MII signals when an error is inserted during transmission.

When receiving, the receive data valid signal indicates valid reception (Figure 14). When this signal is de-asserted reception has ended and the MAC ensures that the frame is valid by way of the FCS. If the receive error signal is asserted (Figure 15), then the MAC drops the packet and returns to the idle state.

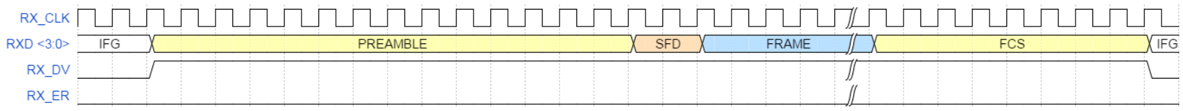


Figure 14 - MII - Receive - Normal Operation

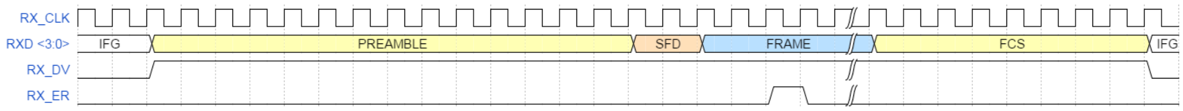


Figure 15 - MII - Receive - Receive Error

2.2.3.2 Reduced Media Independent Interface

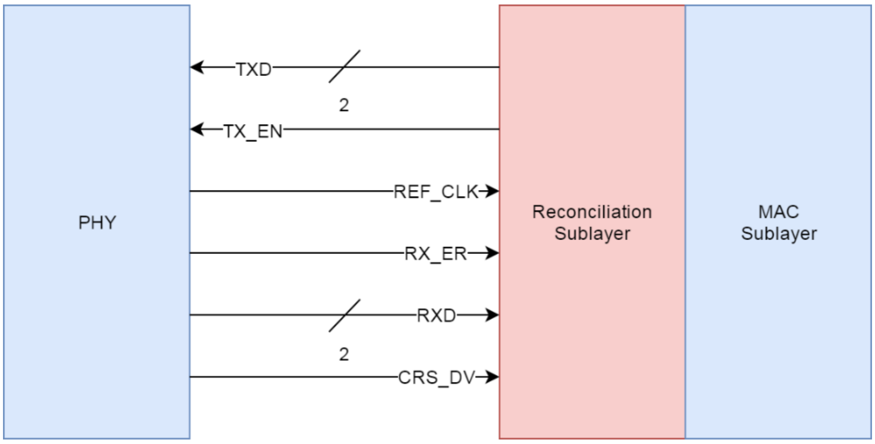


Figure 16 - RMII signals

The Reduced Media Independent Interface (RMII) (Figure 16) was designed to reduce the number of signals between the MAC and PHY, originally requiring 16 signals, with the RMII only requiring 8 signals, but with some loss of functionality [11].

This is achieved by halving the size of the data buses and increasing the reference clock speed to 50MHz for both 100 Mb/s and 10 Mb/s, with the PHY taking care of slowing the data down for transmission at the lower speed. The figures below show the same conditions as with the MII to allow for comparison between the two interfaces.

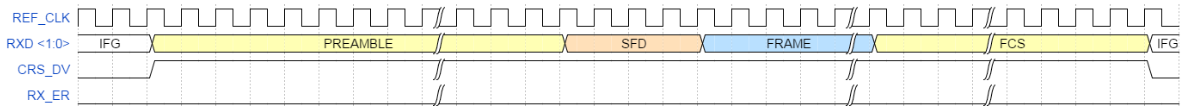


Figure 17 - RMII - Receive - Normal Operation

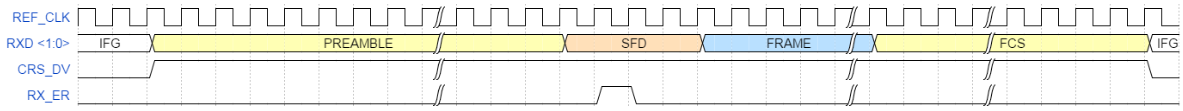


Figure 18 - RMII - Receive - Receive Error

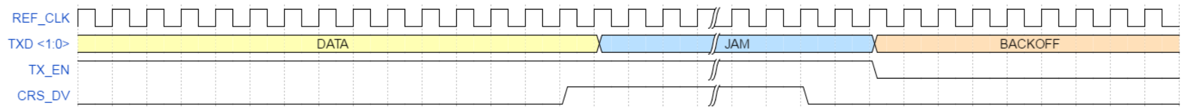


Figure 19 - RMII - Transmit - Collision

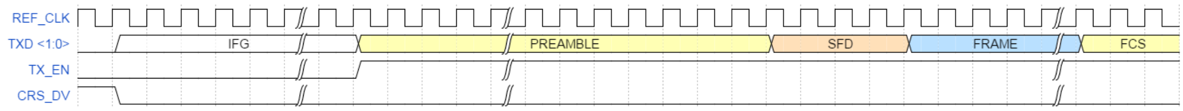


Figure 20 - RMII - Transmit - Deferring

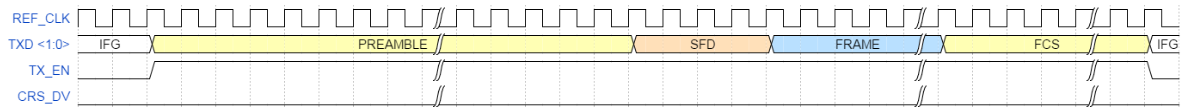


Figure 21 - RMII - Transmit - Normal Operation

2.2.3.3 Gigabit Media Independent Interface

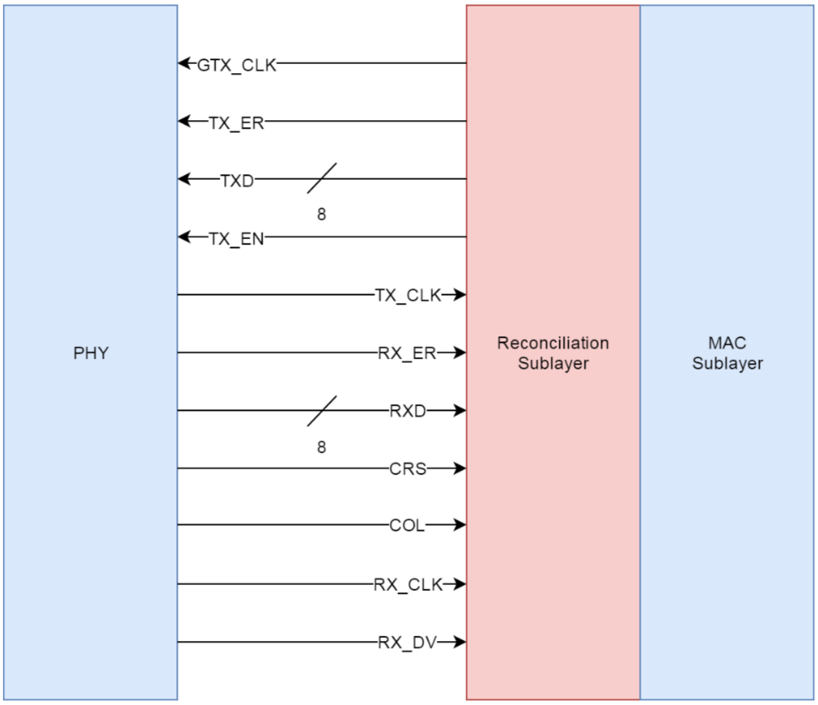


Figure 22 - GMII signals

The Gigabit Media Independent Interface (GMII) [9] (Figure 22) is backwards compatible with the original MII, but doubles the number of data lines for both transmit and receive, and adds an additional Gigabit Transmit (GTX) clock for transmission at 1000 Mb/s which is produced by the MAC sublayer. When the PHY indicates that it is operating at the faster speed, the GMII interface is used as shown in Figure 25 and Figure 26. At the slower speed the MII interface is used as shown in Figure 27 and Figure 28. Receive operation is the same for all speeds (Figure 23 and Figure 24).

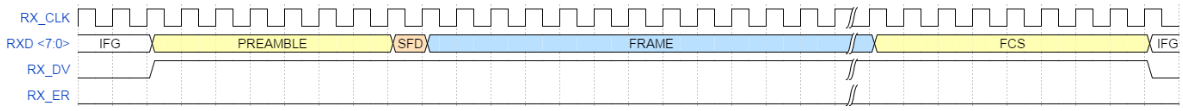


Figure 23 - GMII - Receive - Normal Operation

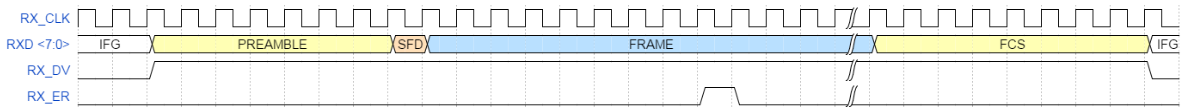


Figure 24 - GMII - Receive - Receive Error

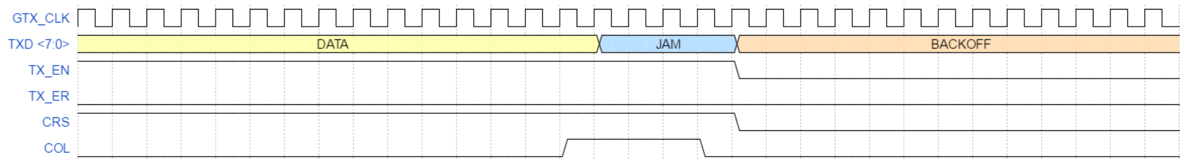


Figure 25 - GMII - Transmit – Collision – 1000Mbps

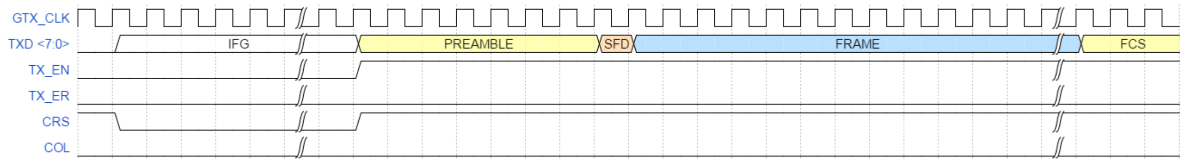


Figure 26 - GMII - Transmit – Deferring – 1000Mbps

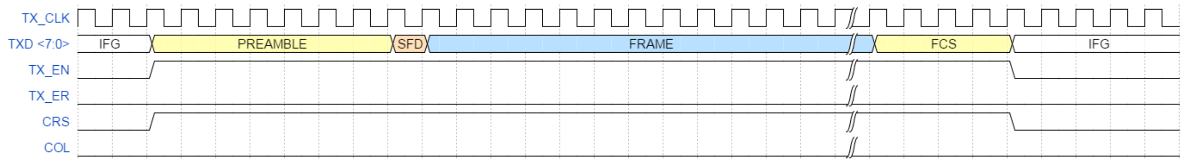


Figure 27 - GMII - Transmit - Normal Operation – 100Mbps

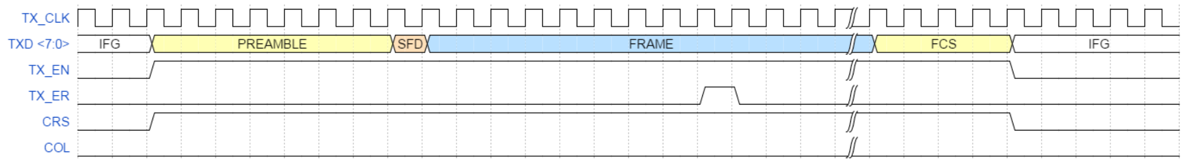


Figure 28 - GMII - Transmit - Transmit Error – 100Mbps

2.2.3.4 Reduced Gigabit Media Independent Interface

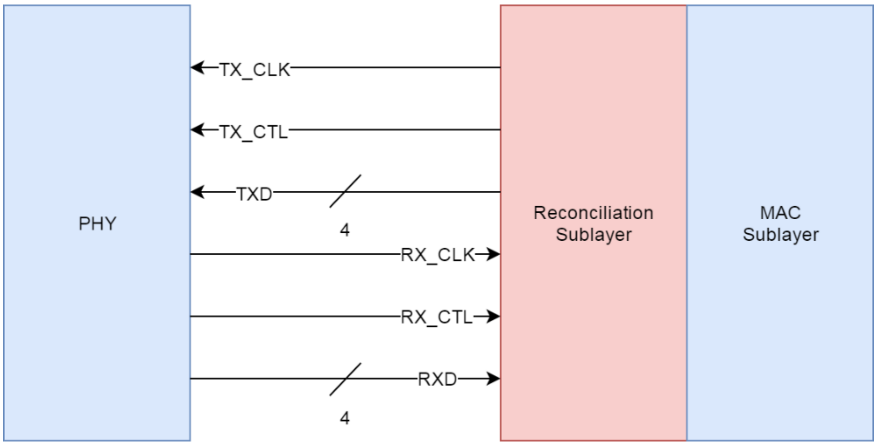


Figure 29 - RGMII signals

The Reduced Gigabit Media Independent Interface (RGMII) [12](Figure 29) is a version of the GMII which halves the number of data lines and multiplexes the control signals to and from the PHY, ending up at 12 signals. The data and control signals are available on both edges of the clock, effectively allowing 8 data signals each for transmit and receive, as in GMII plus 4 control signals. TX_EN is presented on the rising edge of TX_CLK on TX_CTL and TX_EN XOR TX_ER is presented on the falling edge (Figure 33 and Figure 34). RX_DV and RX_ER share RX_CTL in the same way with respect to RX_CLK (Figure 30 and Figure 31). If RX_DV is asserted, this is equivalent to CRS and will be de-asserted during transmission unless there is a collision, effectively replicating the COL signal (Figure 32).

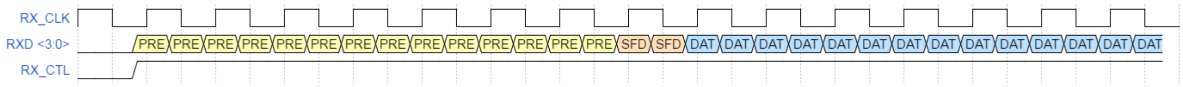

RX_CLK
RXD <3:0>
RX_CTL
PRE PRE PRE PRE PRE PRE PRE PRE PRE PRE PRE PRE PRE PRE SFD SFD DAT DAT DAT DAT DAT DAT DAT DAT DAT DAT DAT DAT DAT DAT

Figure 30 - RGMII - Receive - Normal Operation

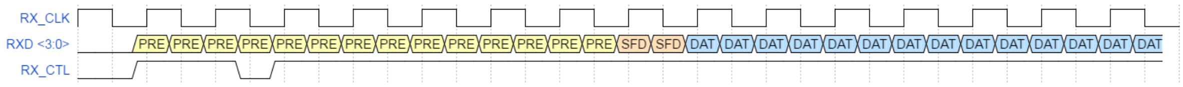

RX_CLK
RXD <3:0>
RX_CTL
PRE PRE PRE PRE PRE PRE PRE PRE PRE PRE PRE PRE PRE PRE SFD SFD DAT DAT DAT DAT DAT DAT DAT DAT DAT DAT DAT DAT DAT DAT

Figure 31 - RGMII - Receive - Receive Error

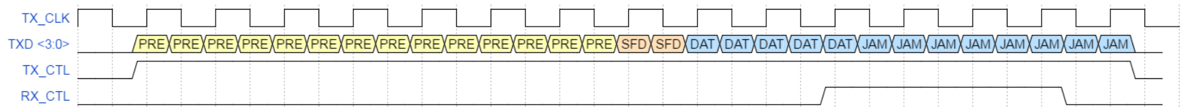

TX_CLK
TXD <3:0>
TX_CTL
RX_CTL
PRE PRE PRE PRE PRE PRE PRE PRE PRE PRE PRE PRE PRE PRE SFD SFD DAT DAT DAT DAT DAT JAM JAM JAM JAM JAM JAM JAM JAM

Figure 32 - RGMII - Transmit - Collision

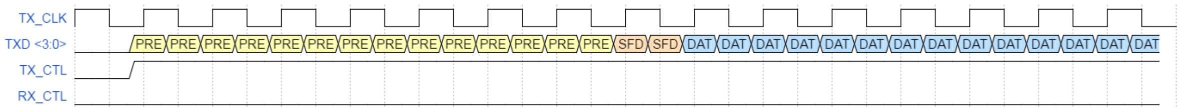


Figure 33 - RGMII - Transmit - Normal Operation

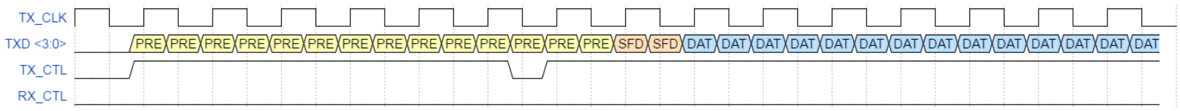


Figure 34 - RGMII - Transmit - Transmit Error

Because it is Double Data Rate (DDR), the diagrams above make each nibble and appropriate clock edge apparent.

2.2.3.5 Station Management

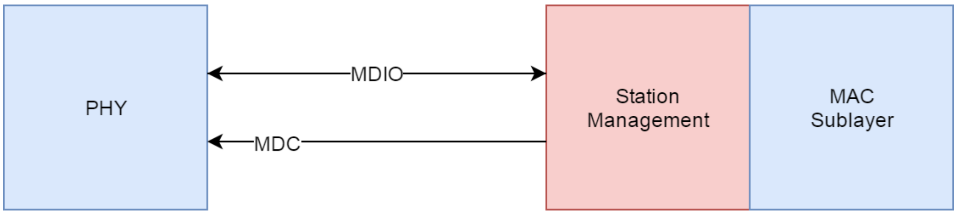


Figure 35 - Station management signals

Station management is carried out by the station management entity, it connects to one or more PHYs using the Management Data Input/Output (MDIO) bus. There are two signals, the data (MDIO) and the Management Data Clock (MDC). The MDIO bus connects to multiple addressable PHYs from which information can be gathered. PHY registers contain status information about the PHY and can also be used to control the PHY. There can be up to 32 registers available on a PHY; Table 2 shows some of the most relevant registers and their addresses [9].

Table 2

Register Address	Name
0	Control
1	Status
2	PHY Identifier
15	Extended

The control register is used to perform actions on the PHY such as reset, set the speed, etc. The register information can be found in clause 22 of IEEE 802.3.

The MDIO bus can be thought of as a hybrid of Ethernet and I2C. It is a 2-wire multi-master addressable bus protocol where devices and then registers on the devices are addressed and written to/read from like I2C, but the frame format contains a preamble and start of frame delimiter as in Ethernet. The preamble consists of 32 consecutive 1s, Figure 36 shows the rest of the MDIO frame.

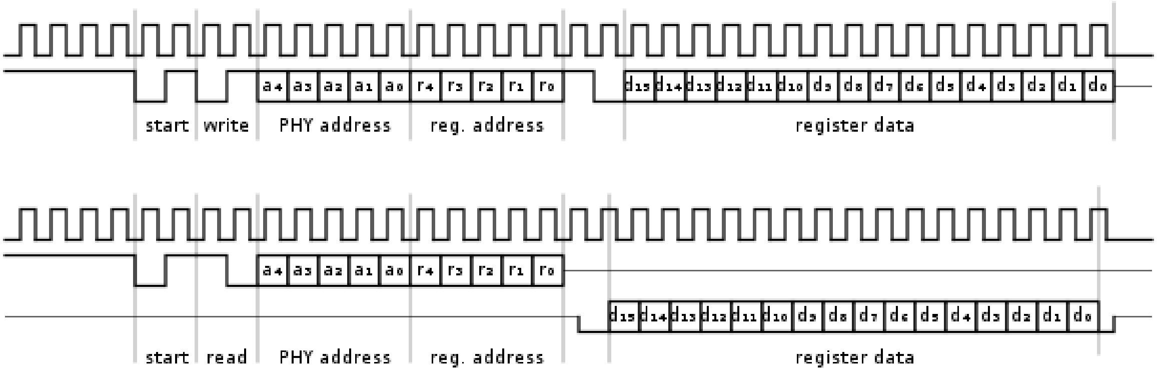


Figure 36 - MDIO - Writing (top) & reading (bottom) [13]

The PHY address and register addresses are both 5 bits wide, allowing for up to 32 PHYs with 32 16-bit registers each.

During a write, the station management entity remains the master of the bus for the duration of the frame, this is signalled using the 2-bit opcode field immediately preceding the PHY address. An opcode of 0b10 is used for reading and a value of 0b01 is used for writing. A high impedance is presented by the station management entity during the 2-bit turnaround time (after the register address) for reading and the PHY takes control of the MDIO bus and transmits the contents of the appropriate register. During a write operation the value 0b10 is written by the station management entity during the turnaround time followed by the 16-bits of data to be written [9].

2.3 Field Programmable Gate Arrays

A Field Programmable Gate Array (FPGA) is a highly configurable logic device allowing Hardware Description Language (HDL) to describe digital logic from which a working design can be synthesised and eventually implemented on the device. An FPGA is built up of a multitude of configurable logic blocks which are all interconnected via a configurable fabric of internal connections.

2.3.1 Configurable Logic Block

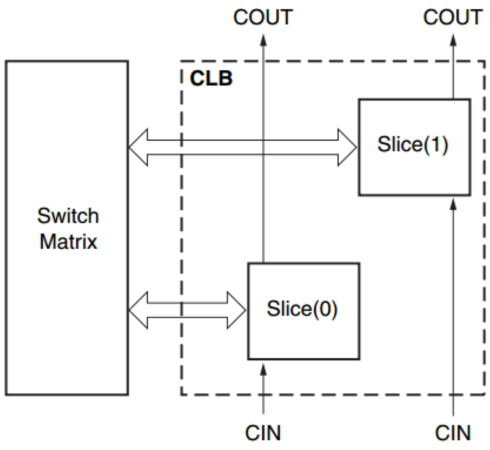


Figure 37 - Configurable Logic Block [14]

The FPGA is a collection of Configurable Logic Blocks (CLB) which are connected to each other via a switch matrix (see Figure 37). Each CLB can contain a number of internal logic blocks sometimes known as slices [14]. The figure below shows one of the possible internal configurations of a slice.

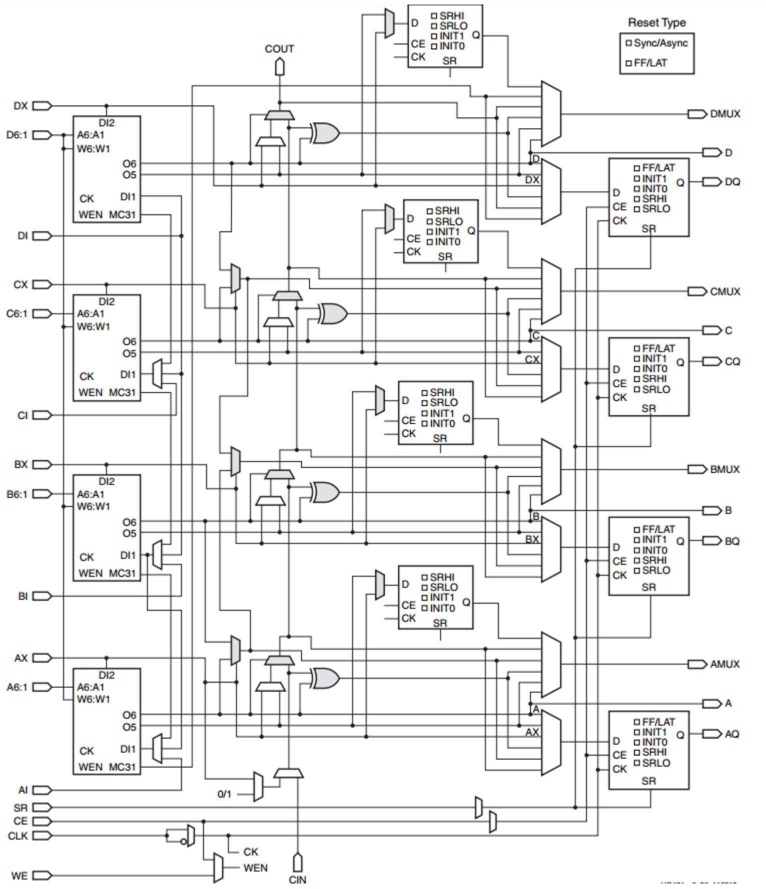


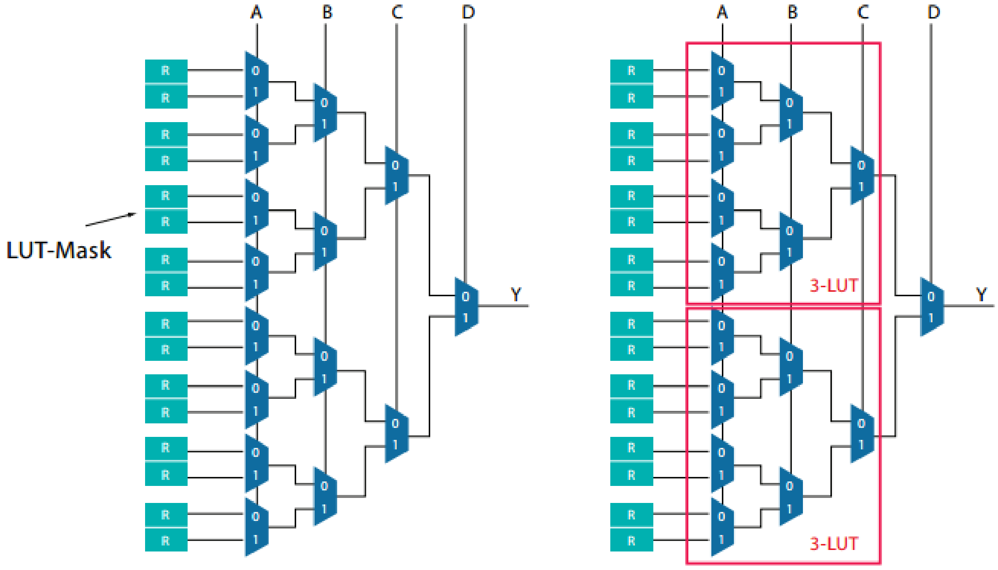
Figure 38 – Slice [14]

A slice is usually built up of a few primitives:

- Look Up Tables.
- D Flip Flops.
- Multiplexers.

2.3.1.1 Look Up Tables

A Look Up Table (LUT) is a multiple input, single output combinatorial logic block and allows for the multiple input signals to map to an output based on a logic equation which generates a truth table for the device and implements that specific logic function. When the FPGA is being configured, the LUT's table gets filled out with what is known as the LUT mask (see Figure 39). The LUT is the heart of most FPGA operations and usually is available in 5 or 6 input varieties on-chip.



a'b'c'd' + abcd + abc'd' = 1000 0000 0000 1001 = 0x8009

Figure 39 - Internal Design of a LUT [15]

2.3.1.2 D Flip-Flop

The D Flip-Flop (FF) is the memory unit present in most CLBs. At the rising edge of the clock, the data present on the D input is captured and is presented on the Q output of the device. As such it is capable of holding a single bit of information. A series of D FFs is called a register, and if data moves from one FF to the next in a line, this is known as a shift register. This D FF is shown in Figure 40.

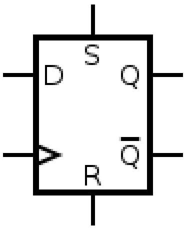


Figure 40 - D Flip-Flop [16]

2.3.1.3 Multiplexer

A multiplexer or mux is a device used to select between signals, for example a 2 to 1 mux will have a control signal that when high will pass the first input to the output and when low will pass the second to the output. Figure 41 shows the diagram for a 2 to 1 mux.

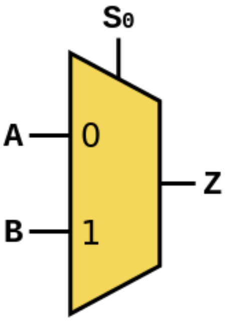


Figure 41 - 2 to 1 Multiplexer [17]

3 Design

3.1 Overview

As mentioned in the previous section, there are a few prerequisites for implementing a MAC sublayer; namely station management and the reconciliation sublayer. The design of this implementation is shown in Figure 42.

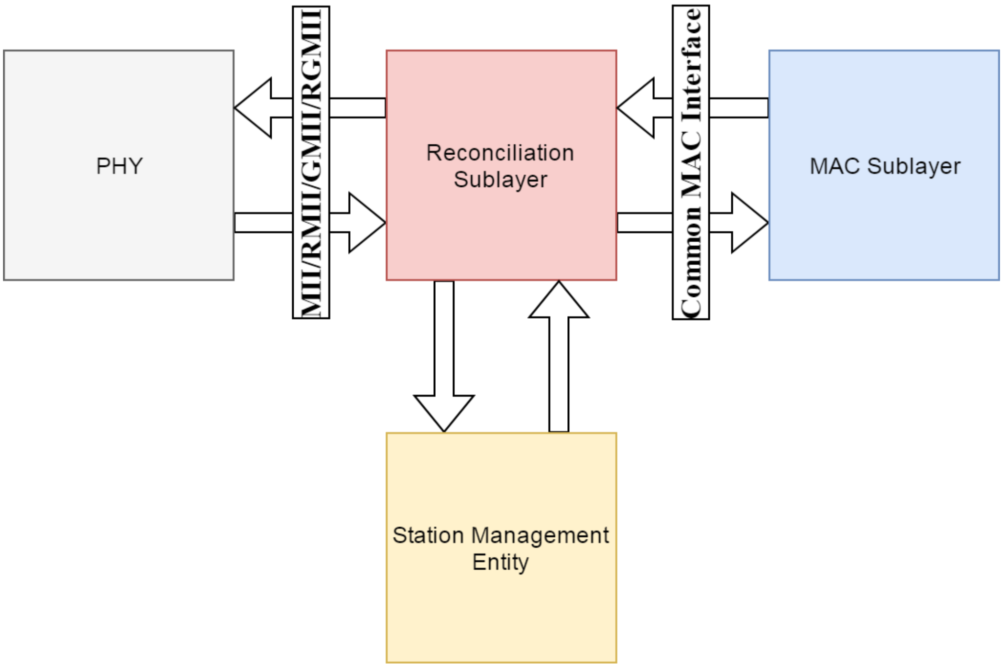


Figure 42 - Overall Block Design - Open Source Ethernet MAC

As you can see from the diagram, the interface to the PHY is using one of the aforementioned MIIs. From a portability point of view it makes sense to develop the different interfaces, so that the design can be moved and implemented easily on other devices which may have different PHY interfaces.

3.2 MAC Sublayer

The MAC sublayer has been talked about extensively in the previous section, the details of which will only be summarised where necessary in this section. For more information, please refer back to section 2.2.

The design of the MAC sublayer core is available in Figure 43. The arrows show the path of the data between the reconciliation sublayer and the FIFO interfaces for both transmit and receive. The core components are laid out for both transmit and receive and will be further explained in the sections below.

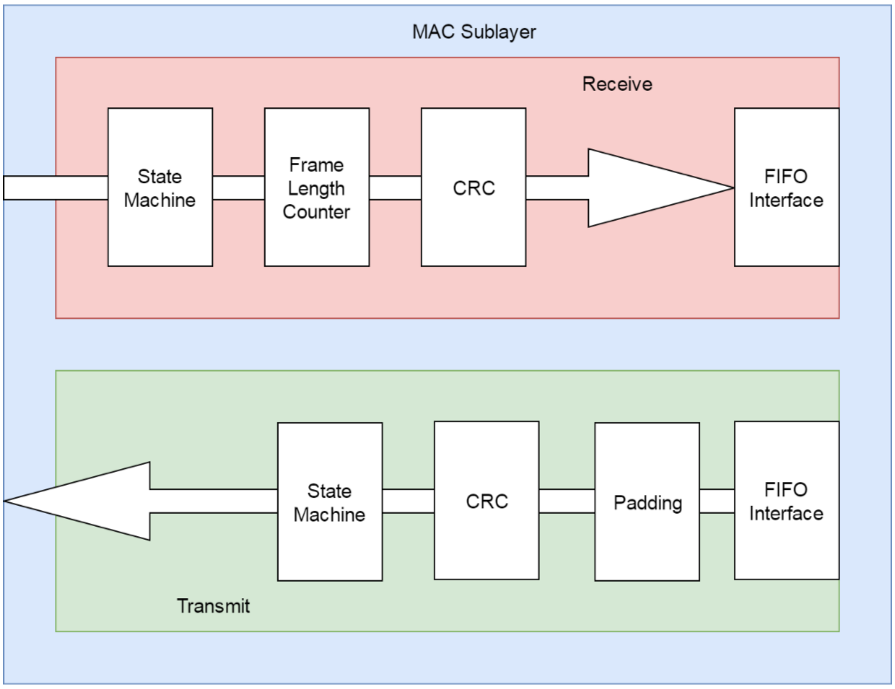


Figure 43 - Block Design - MAC Sublayer

3.2.1 Receive

The receive core of the MAC is primarily made up of a state machine (with states that match the section of the packet that is currently being received) and a 32-bit parallel CRC core. As data is received the state transitions ultimately to either an error or an OK state indicating the validity of the packet. The error is raised to the MAC client, but the packet is passed on through for the client to decide what to do with it. This is by design; as this core is intended for uses in an Ethernet testing unit, both valid and invalid frames carry useful information about the state of the DUT, for example, if the DUT is incorrectly calculating the FCS or is producing packets with the wrong bit order, etc. The data width for receive and transmit data in this MAC is designed to be 1 octet wide, removing the possibility for alignment errors and non-integer numbers of octets.

3.2.1.1 State Machine

A simplified diagram of the states and transitions of the receive core can be seen in Figure 44. The states are described in more detail throughout this section.

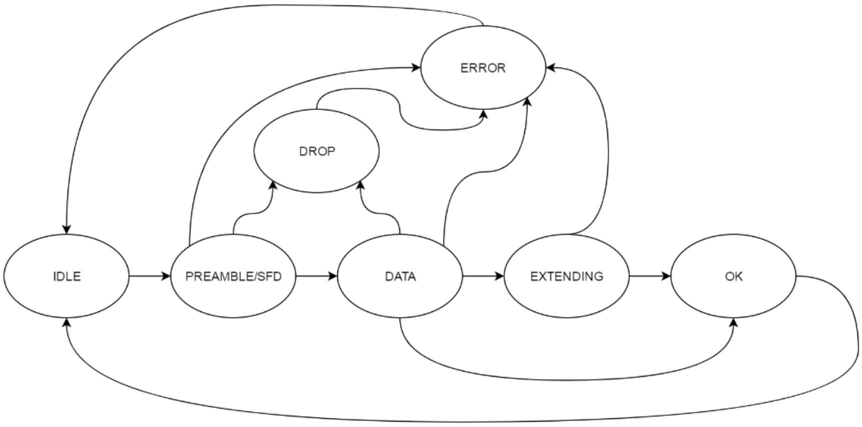


Figure 44 - Simplified state machine diagram

Figure 45 shows the flows when in the idle state. The flow diagram starts at the positive edge of the received clock from the reconciliation sublayer and reaches the terminator in the flow before the next positive edge. At every clock edge in the idle state, receive data valid is sampled, if the data is valid, this indicates the start of a packet. The data itself is then sampled, if the data is preamble (0x55) then the state transitions to the preamble state. This state serves to prepare the MAC for reception of data, and also to mark the start of the frame. On entrance to this state, the CRC module is initialised and on leaving this state it is enabled, to allow calculation of the CRC residue in situ.

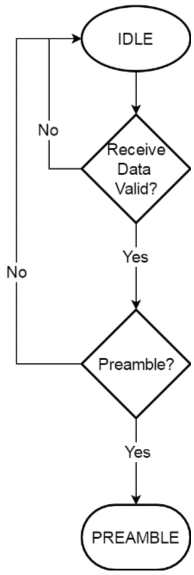


Figure 45 - Flow Diagram - Idle State

When in the preamble state, receive data valid is again sampled. In this case, if the signal is de-asserted, then the state transitions into the error state. If an error is received, then the state moves into the drop state to drop the rest of the packet. If after passing the control signal section (RX_DV and RX_ER), the data is the SFD (0xD5) then the state moves to the data state; if the data is another preamble octet, then the state remains in preamble. Any data octet other than 0x55 or 0xD5 is will cause the packet to be dropped. Figure 46 shows this process.

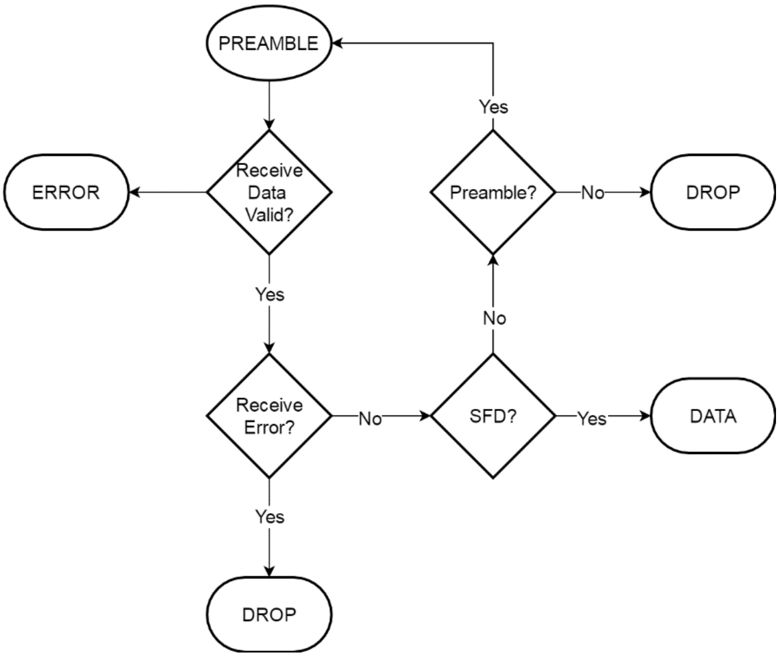


Figure 46 - Flow Diagram - Preamble State

The data state's flow diagram is slightly more complex than any of the other states', however the flow diagram makes understanding the design much easier. This is shown in Figure 47. The conditions to remain in the data state are simply that the data is valid, there are no errors, the packet isn't too long and there is remaining empty memory in the FIFO which the MAC receive side is connected to; this constitutes the top half of the flow diagram.

If the data valid signal has been de-asserted, this indicates that the end of the packet has been reached. The length is checked and if it doesn't lie between the limits mentioned in section 2.2.2.1.4 then the state machine moves to the error state. If the length is within the limits, then the CRC residue is checked (see the CRC section below for more information on how this works), if the residue matches and we are in full duplex mode, we move to state

OK to indicate correct reception of a packet, the slot timer is checked prior to this if in half duplex mode. If the timer has expired, then the state changes to OK; we move to the extend state in the case that the carrier is being extended; with no carrier we transition to the error state.

On entry to the data state, the FIFO enable signal goes high and, for the first octet, the start of packet signal is raised.

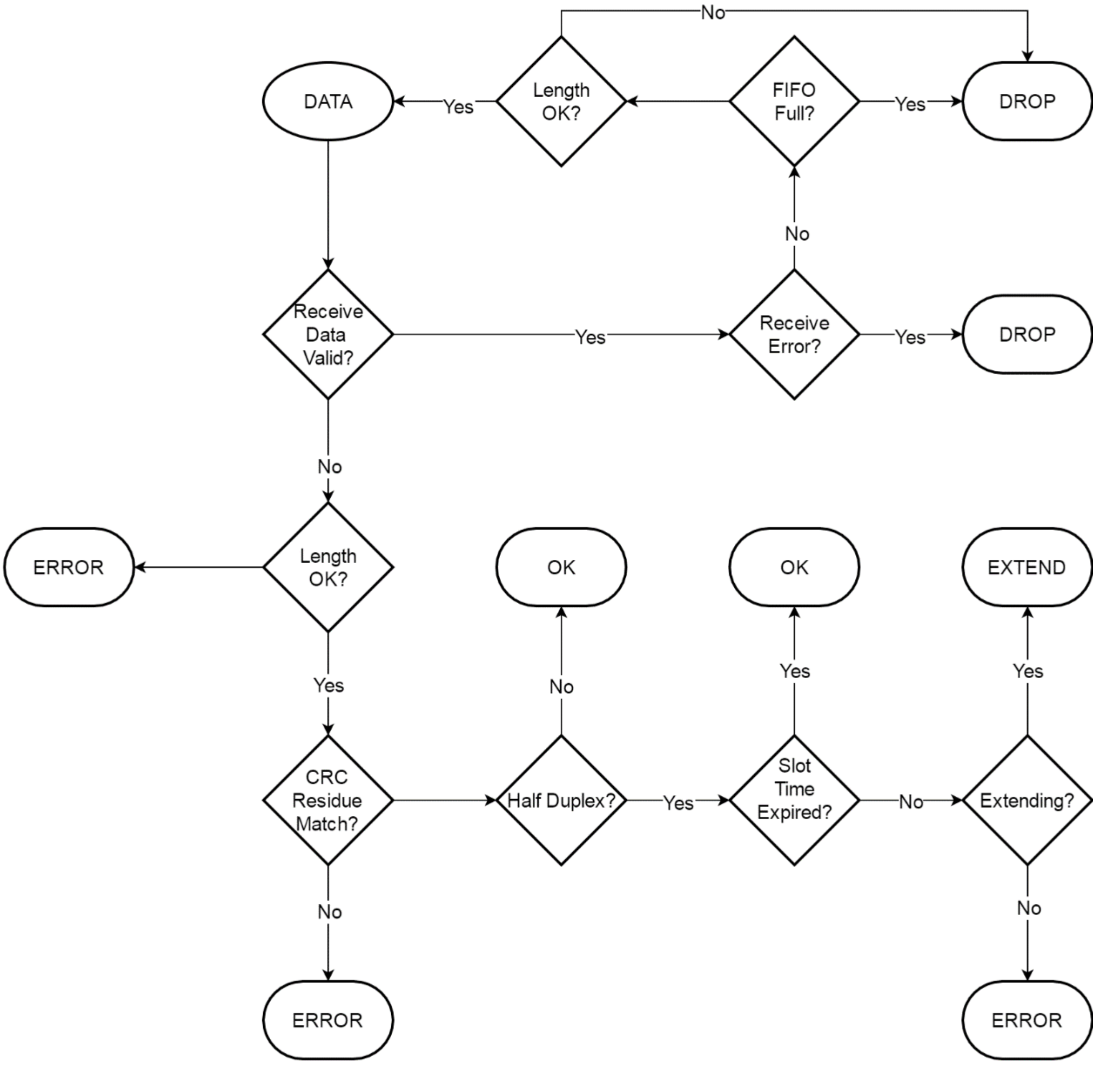


Figure 47 - Flow Diagram - Data State

If the extending signal is high at the positive receive clock edge, then we remain in the extend state. If the extend signal is low, and the slot time has expired, we move to an OK state, otherwise, we move to the error state due to the slot time not being reached. The flow for this state is shown in Figure 48.

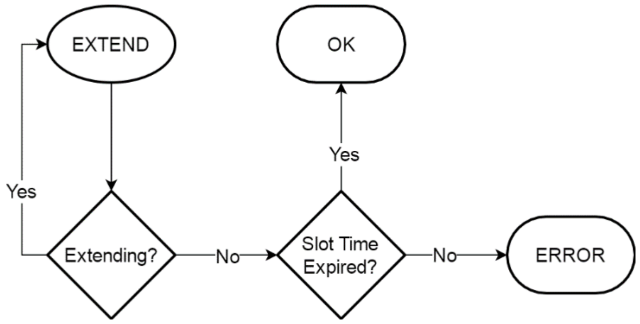


Figure 48 - Flow Diagram - Extend State

The other states are simple enough to not require a flow diagram. In the drop state, the MAC can optionally stop writing to the FIFO in order to drop the packet, and remains in that state until receive data valid is de-asserted. At this point there is a state change to error.

In the error state, a signal is raised to the MAC client that there was an error and to either ignore or mark the last received packet.

In the OK state, the packet is counted as valid and the next state is set to idle.

The last octet written to the FIFO is accompanied by the end of packet signal.

3.2.1.2 Full Duplex

As can be seen from the above flow diagrams, operation in full duplex is simpler than half duplex. The extend state is not present, and there is no regard for the slot time. The design allows for the half duplex functionality to be removed via a parameter if it is not needed.

3.2.1.3 Half Duplex

3.2.1.3.1 Bursting

Packet bursts will be accurately received by this core. The state machine will return to state idle within two clock cycles of either the end of a packet, or the end of carrier extension. Allowing it to transition to the preamble state again. The preamble is 7 octets long meaning that there is ample time to receive the next packet of a burst.

3.2.1.3.2 IFG Shrinkage & Short Preamble

As the core does not count how long the IFG or the preamble are, it will not be affected by phenomena such as:

- IFG shrinkage
    - IFG can shrink from 12 octets to 8 octets.

- Short Preamble
    - The PHY can miss the first few preamble bits due to use of Ethernet beyond its standard constraints e.g. cat 6 cable longer than 100m, etc.

3.2.1.4 Counters

A few counters have been included in the design to report certain useful metrics, such as:

- Valid packets
    - Count of the number of times the OK state has been reached.
- Short packets
    - Number of times the error state has been reached and the too short signal has been high.
- Long packets
    - Number of times the error state has been reached and the too long signal has been high.
- Errored packets
    - Number of times the error state has been reached.
- Dropped packets
    - Number of times the drop state has been reached.

3.2.2 Transmit

The transmit core, like the receiver, is made up of primarily a state machine and another instance of the same 32-bit parallel CRC core. The state machine transitions are shown in the section below, along with an explanation. Collision and carrier sense signals are only relevant in half duplex mode.

3.2.2.1 State Machine

When the core starts, the machine defaults to the defer state. The job of this state is to defer to other, already transmitting stations by listening to the carrier and waiting for a free medium before changing state. Figure 49 shows the logic for this state. In half duplex mode, the carrier is evaluated and the state transitions to the IFG state when the CRS signal drops. In full duplex mode, the state changes to the IFG state on the next rising edge of the transmit clock, regardless of the state of the carrier.

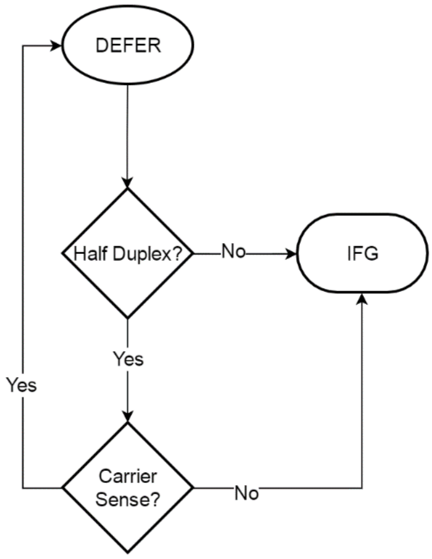


Figure 49 - Flow Diagram - Transmit - Defer State

The IFG state exists to enforce an IFG between subsequent packets; this design currently has no provision for packet bursting, as it is usually turned off for DUTs in Asidua. However, it wouldn't be difficult to extend the current state machine to accommodate packet bursting; logic would have to be included at the FIFO interface to determine whether to carry out a burst or not, though this would not represent significant development time.

If CRS is asserted during the IFG period, the state machine will return to the defer state. On expiry of the IFG timer, the state will change to idle. CRS will be ignored if in full duplex mode, as there is no contention for the medium. The flow is shown in Figure 50.

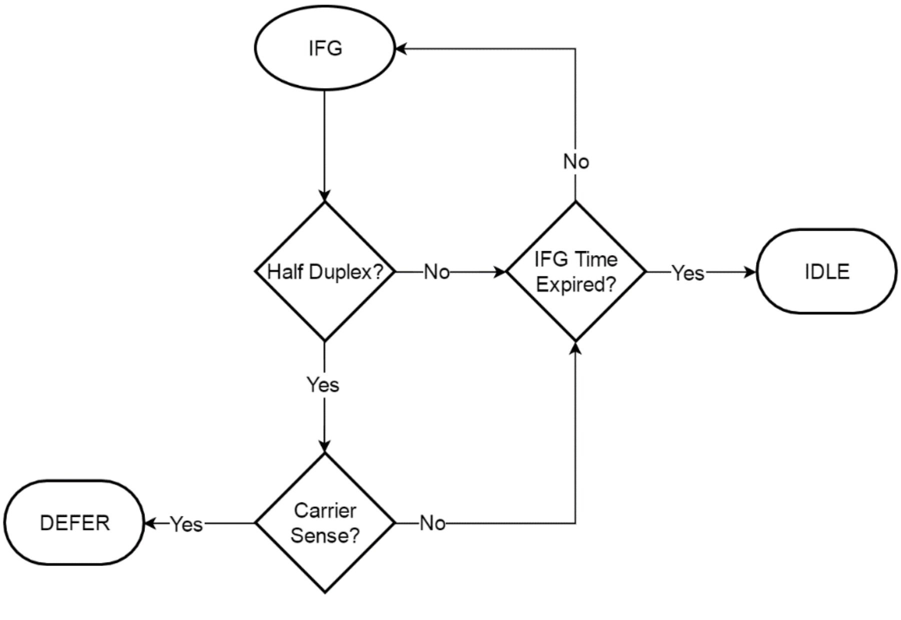


Figure 50 - Flow Diagram - Transmit - IFG State

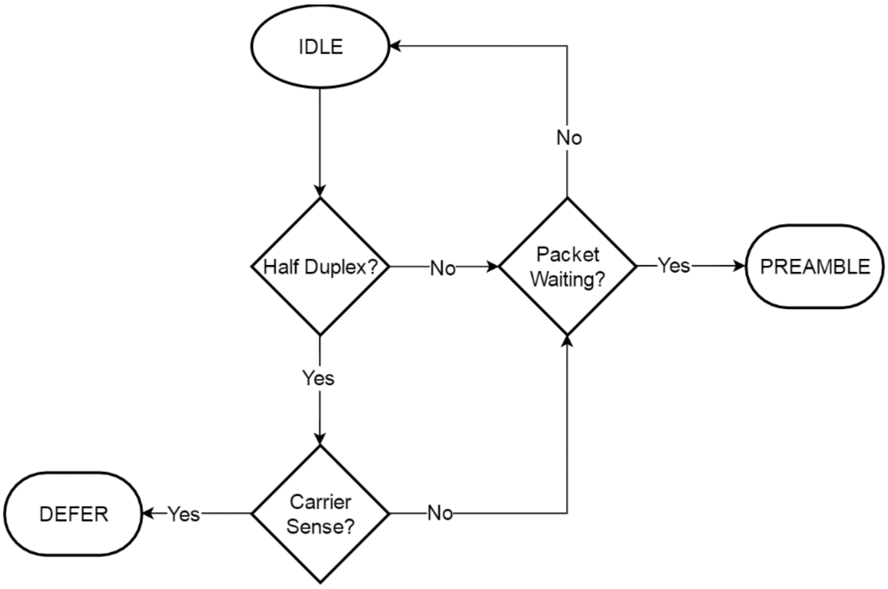


Figure 51 - Flow Diagram - Transmit - Idle State

Figure 51 shows the flow diagram when in the idle state. When in idle, there is no data on the wire and the MAC is ready to send a packet. The state machine remains at idle until either the CRS signal is raised in half duplex or a packet becomes available for transmission. On sensing the carrier in half duplex mode, the state will return to defer.

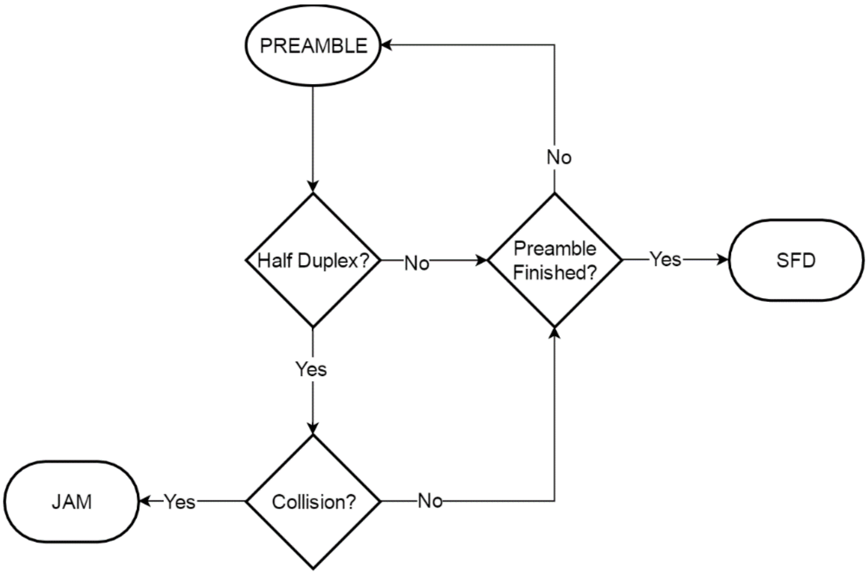


Figure 52 - Flow Diagram - Transmit - Preamble State

In the preamble state (see Figure 52), the MAC transmits 7 octets of preamble, which is determined using a counter. If during this time a collision is detected, then CSMA/CD rules apply and a jamming signal is sent to enforce the collision (half duplex only).

At the end of the preamble the state updates to the SFD state, where the MAC transmits the SFD (0xD5) before transitioning to the data state. Again, if a collision condition is detected in half duplex mode, the next state becomes the jam state.

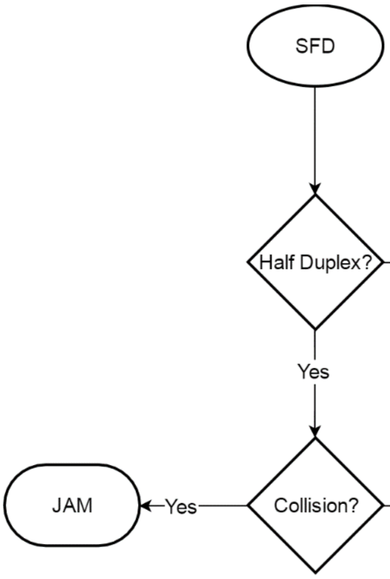


Figure 53 - Flow Diagram - Transmit - SFD State

The data state has the same outcome for collisions as the previous two states. The state remains in state data until the end of frame signal is seen at the FIFO interface at which point it will be determined if padding is required i.e. frame length less than 64 bytes. The state transitions to state pad if this is the case, otherwise the state moves to state FCS. On the transition to this state, the CRC module is initialised and enabled so that the outgoing data is passed to the module to calculate the CRC on the fly.

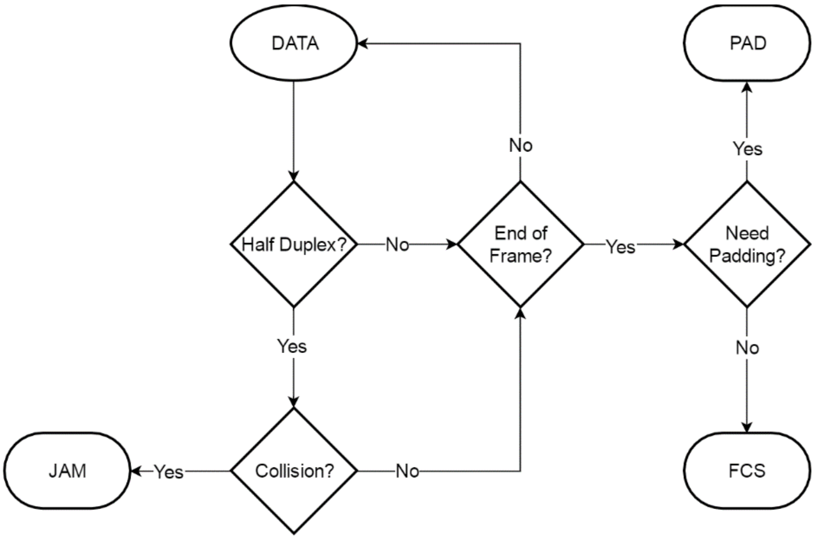


Figure 54 - Flow Diagram - Transmit - Data State

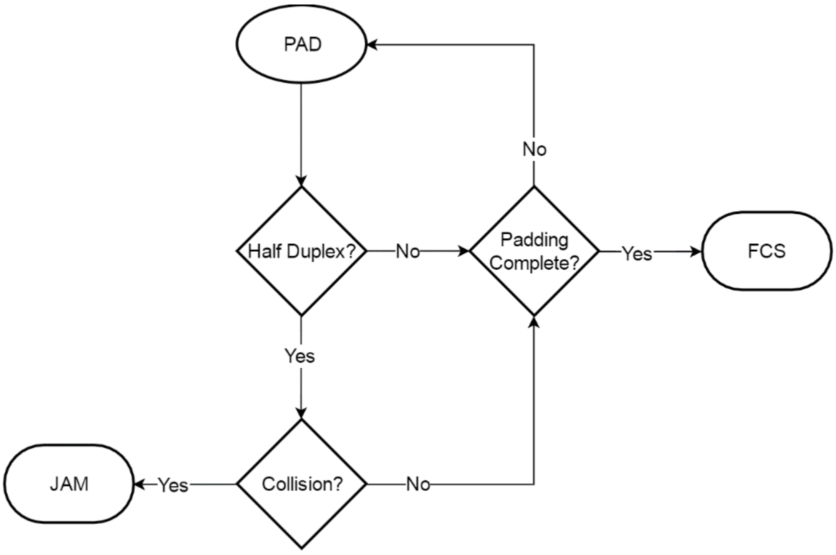


Figure 55 - Flow Diagram - Transmit - Pad State

In the pad state, a counter for the amount of padding decrements and when the padding is complete, the state transitions to the FCS state.

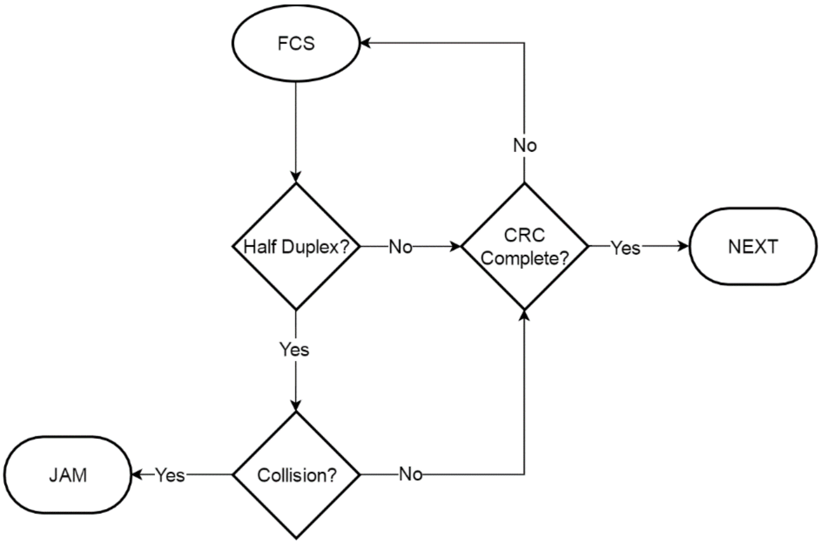


Figure 56 - Flow Diagram - Transmit - FCS State

In the FCS state, the 32-bits of the CRC are shifted out onto the transmit data lines. At the transition to this state, the CRC module is disabled and no longer updates the FCS value with the outgoing data, ensuring that the CRC data itself doesn’t loop back through the module.

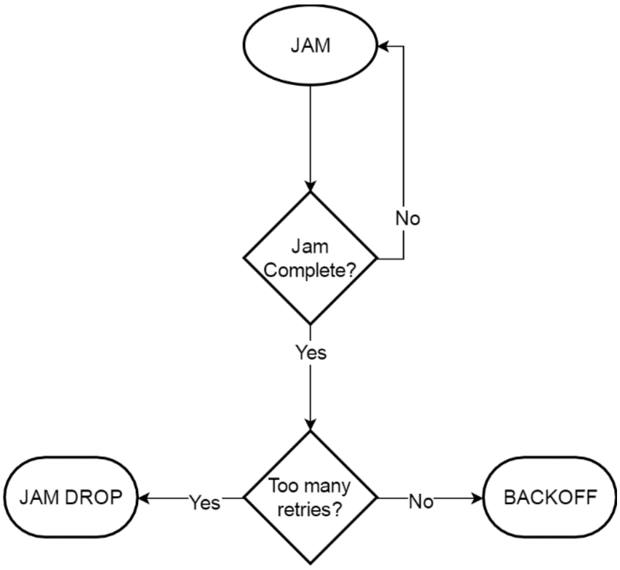


Figure 57 - Flow Diagram - Transmit - Jam State

The jam state is entered whenever a collision is detected. It serves to enforce a collision by transmitting 32-bits of data, therefore elongating the collision and ensuring that other stations on the network will have been made aware of the collision. After the jam the state changes to backoff, unless the number of retries has exceeded 10. Figure 57 shows the logic for this state.

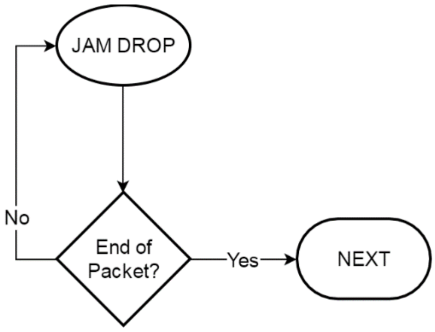


Figure 58 - Flow Diagram - Transmit - Jam Drop State

After the maximum number of retries have been reached, the state updates to state jam drop (see Figure 58), where the packet is read out of the FIFO without raising the retry signal, causing transmission of that packet to be dropped.

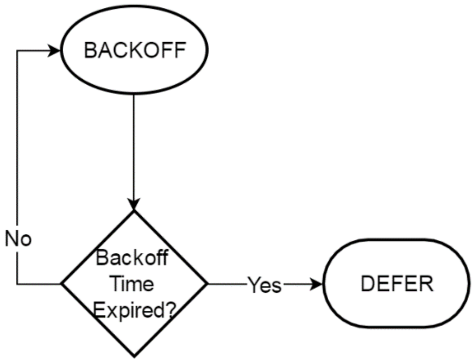


Figure 59 - Flow Diagram - Transmit - Backoff State

The backoff state simply waits for the backoff module to raise the backoff timer expired signal before it updates the state to the defer state where the packet that experienced the collision will be retried.

The next state simply indicates that the MAC is moving to transmit the next packet and is used to count the number of valid transmitted packets.

3.2.2.2 Full Duplex

As with the receive core, operation in full duplex is much simpler and results in a reduction of states and modules: the backoff state and module are not required, the jam and jam drop states are not required, and the state transition logic is greatly simplified.

3.2.2.3 Half Duplex

3.2.2.3.1 Bursting

This core is currently incapable of transmitting a packet burst, however, it would not be difficult to implement such a feature.

3.2.2.3.2 Carrier Extension

Carrier extension is not currently available for this core, there was no way to physically test this with equipment in the DS lab, and therefore it has been omitted in this version of the MAC core. It will be implemented in the next version of the MAC.

3.2.2.4 Counters

The transmit core records a number of values:

- Packets attempted
    - Counted every time the preamble state is entered.
- Packets transmitted successfully

- Number of times the next state has been reached, except when transitioning from jam drop.

3.2.3 CRC

The Cyclic Redundancy Check is used for error detection in many digital communications standards, for example, Universal Serial Bus (USB) uses a 5-bit CRC to error detect transmission errors. Ethernet uses a 32-bit CRC with a polynomial of 0x04C11DB7, which corresponds to the polynomial described in section 2.2.2.1.6. CRC itself is not the subject of this report and it is assumed that the reader is familiar with the concept. For the examples below, USB's CRC has been used to explain the concepts as it is much shorter than the 32-bit CRC of Ethernet, and removes any extraneous calculations.

3.2.3.1 CRC Calculation – Serial

CRCs are usually implemented using a Linear Feedback Shift Register (LFSR). Data is shifted in one bit at a time and the register contains the current CRC value. Depending on the generating polynomial, some flip-flops within the register have an XOR gate between them and the next, causing the data to change as it moves through the shift register. This can be seen in Figure 60 which shows the CRC LFSR circuit for USB. The 5 flip-flops contain the current CRC value, which changes as data is clocked into the register.

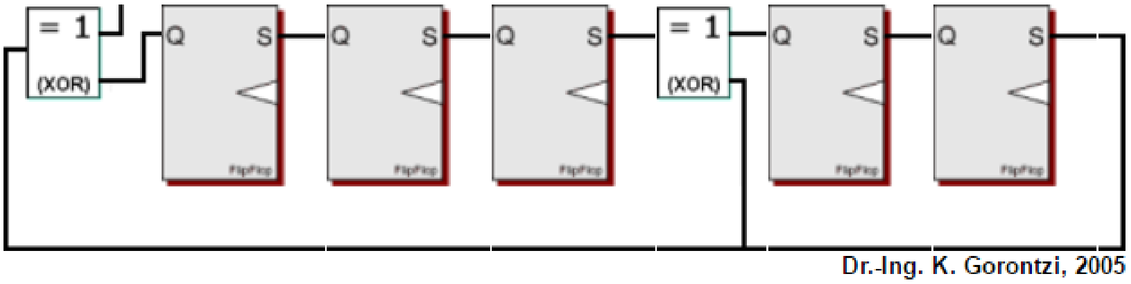


Figure 60 - USB CRC LFSR [18]

USB's generating polynomial is 0x5 or 0b101, corresponding to an XOR between the output of the most significant bit in the shift register and the data input (bit 0 of the polynomial, the feedback portion of the LFSR) and an XOR between this value and the output of the second flip-flop in the register (from left to right). Each bit of the CRC can be calculated as follows:

$$CRC_t[0] = CRC_{t-1}[4] \oplus D$$ Equation 6

$$CRC_t[1] = CRC_{t-1}[0]$$ Equation 7

$$CRC_t[2] = CRC_{t-1}[1] \oplus CRC_{t-1}[4] \oplus D$$ Equation 8

$$CRC_t[3] = CRC_{t-1}[2]$$ Equation 9

$$CRC_t[4] = CRC_{t-1}[3]$$ Equation 10

3.2.3.2 CRC Calculation – Parallel

There are a number of sequential algorithms for creating logic that performs CRC in parallel for a particular data width. For example, USB's CRC can be calculated in parallel for a data path of 4 bits, allowing for the CRC to be evaluated 4 bits at a time.

$$\begin{aligned} CRC_t[0] = {} & CRC_{t-1}[1] \oplus CRC_{t-1}[4] \\ & \oplus D[0] \oplus D[3] \end{aligned}$$ Equation 11

$$CRC_t[1] = CRC_{t-1}[2] \oplus D[1]$$ Equation 12

$$\begin{aligned} CRC_t[2] = {} & CRC_{t-1}[1] \oplus CRC_{t-1}[3] \\ & \oplus CRC_{t-1}[4] \oplus D[0] \\ & \oplus D[2] \oplus D[3] \end{aligned}$$ Equation 13

$$\begin{aligned} CRC_t[3] = {} & CRC_{t-1}[2] \oplus CRC_{t-1}[4] \\ & \oplus D[1] \oplus D[3] \end{aligned}$$ Equation 14

$$\begin{aligned} CRC_t[4] = {} & CRC_{t-1}[0] \oplus CRC_{t-1}[3] \\ & \oplus D[2] \end{aligned}$$ Equation 15

3.2.3.3 An Algorithm for Calculating CRC in Parallel

Using the information above, one could implement a CRC module for USB with a 4-bit wide data path, but the architecture would be fixed meaning that another core would have to be implemented if another CRC architecture was required.

In light of this, a configurable CRC module was designed. The module uses generate statements to have the synthesiser evaluate the sequential code at synthesis and produce the required core based on parameters, the automatic keyword in the function definition tells the synthesiser that the function is re-entrant, thereby allowing a recursive algorithm.

The pseudocode for this is presented below in Appendix A. The way it works is by recursively stepping back through the register states and ultimately working out the path that the data would have taken if it were passed through the LFSR in serial. The function is called for each bit of the CRC. This process is presented in the spreadsheet available from the GitHub [19], it is too large to present here in its entirety.

Table 3

Data Width	Value in Flip-Flop
1	C3
2	C2
3	C4 ^ D0 ^ C1
4	C3 ^ D1 ^ C0
5	C2 ^ D2 ^ C4 ^ D0
6	C4 ^ D0 ^ C1 ^ D3 ^ C3 ^ D1
7	C3 ^ D1 ^ C0 ^ D4 ^ C2 ^ D2
8	C2 ^ D2 ^ C4 ^ D0 ^ D5 ^ C4 ^ D0 ^ C1 ^ D3

In the table above, the calculation of the most significant bit of the CRC shown relative to the data width. The algorithm in the appendices steps backwards, looking at where the value in the flip-flop came from (the previous state) until the data width reaches 0, at which point the calculation for that bit of the CRC is complete.

3.2.3.4 Ethernet Frame Check Sequence

The Ethernet FCS has a few constraints which must be adhered to:

- The initial value for the CRC in the calculation must be all 1s.
- Data is transmitted and presented at the MAC sublayer in reverse bit order, therefore data passed to the CRC module must first be reversed.
- As a result, prior to transmission the CRC value out of the core must be reversed as well.
- The Ethernet FCS is transmitted in the opposite byte-order to the rest of the packet and is transmitted in little-endian fashion, meaning that the bytes must also be reversed.
- The Ethernet FCS is inverted as well, as per the standard.

When receiving a frame, the FCS can be validated in situ by running the frame and the received FCS through the CRC module, if the FCS is valid then a residue of 0xC704DD7B will be produced. This is helpful as there is no way to know where the end of the frame is at the time of reception, as the data valid signal remains high until the FCS is received [9].

3.2.4 Backoff

3.2.4.1 Pseudorandom Number Generator

The random number generator is accomplished by using a 10-bit register, setting the value based on the previous state at each clock cycle to achieve a pseudo-random number. As stations will never all be started at exactly the same time, the assumption can be made that the random numbers generated will rarely correlate with each other.

The register is initially set to 0 and the first few outputs in the sequence are noted below.

Table 4

9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	1	1	0
0	0	0	0	0	0	1	1	0	0
0	0	0	0	0	1	1	0	0	1
0	0	0	0	1	1	0	0	1	1
0	0	0	1	1	0	0	1	1	0
0	0	1	1	0	0	1	1	0	0
0	1	1	0	0	1	1	0	0	1
1	1	0	0	1	1	0	0	1	1
1	0	0	1	1	0	0	1	1	0

The register can be considered a similar construction to the LFSR in the CRC section. The number in the register will take 1024 cycles before the pattern will repeat.

The 2^k component of the random calculation is accomplished by using the number of bits that correspond to the number of retries i.e. zero retries, just use bit 0; five retries, use bits 5 to 0.

This number is then decremented every time a slot time expires until it reaches zero, at which point the backoff core sends a trigger signal to move the transmit core out of the backoff state.

3.3 Reconciliation Sublayer

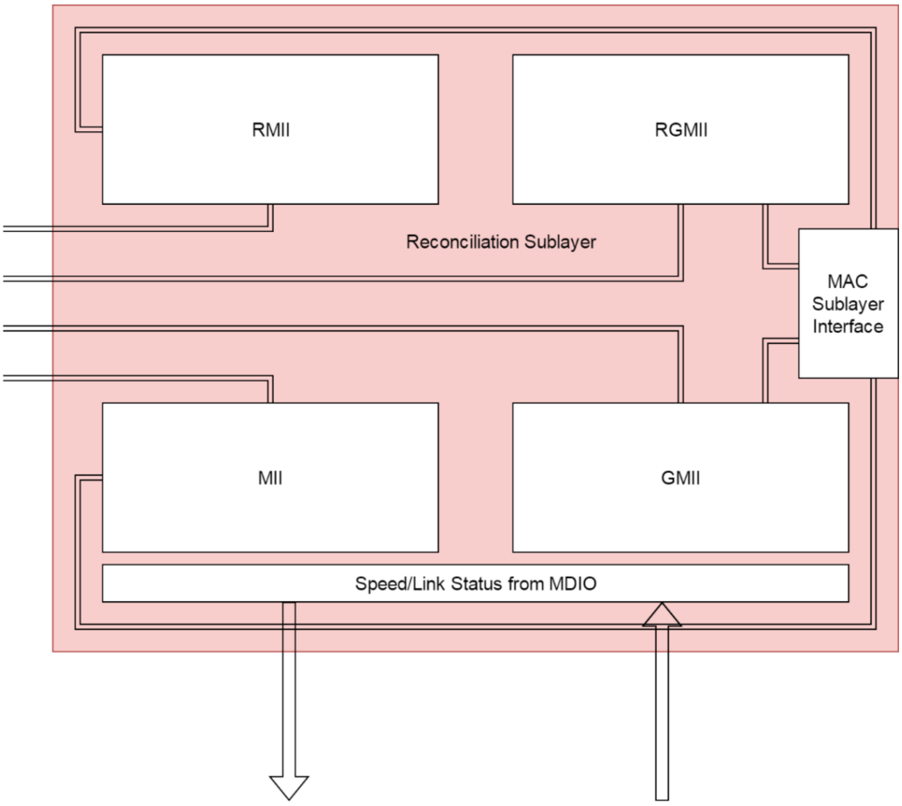


Figure 61 - Reconciliation Sublayer – Diagram

The reconciliation sublayer provides several PHY interfaces in one module. This is controlled via a parameter which will reduce the logic at the time of synthesis. It also connects to the station management core to interrogate the PHY for its speed and link status.

The MIIs provide a simple mapping of PHY signals to an 8-bit data path for both transmit and receive, as well as control signals (RX_DV, RX_ER, TX_EN, TX_ER, COL, CRS). In doing this, the MAC can be designed without regard for the PHY it is to be attached to.

3.3.1 MII

The MII data path is 4-bits wide, as such there are two clock dividers in the MII module which half the clock speed to allow for the 8-bit data path on the MAC interface. Data is shifted into an octet wide register one nibble at a time on the rising edge of the PHY transmit and receive clocks. The divided clocks are passed through to the MAC.

3.3.2 RMII

The RMII data path is only 2 bits wide, and uses a single 50MHz clock. For this, there is a single clock divider which divides the clock speed by 4 and presents this clock as both transmit and receive clocks to the MAC. The COL signal is asserted when both CRS_DV and TX_EN are high. CRS_DV provides both CRS and RX_DV signals to the MAC.

3.3.3 GMII

The GMII module is a simple pass-through, as the MAC signals correlate exactly to the GMII standard. The transmit clock in this case must be produced internally and is input into the RS.

3.3.4 RGMII

DDR operation for outputs is impossible to infer to a synthesiser when writing Verilog, instead the rising and falling signals are made available to be connected to input and output DDR registers when the core is being implemented on a specific device. The first nibble of the 8 bit data buses is connected to the rising signal and the second nibble is connected to the falling signal. Again, the transmit clock is generated internally and passed to both the MAC and the PHY. RX_ER, CRS, RX_DV, COL are all regenerated from the control signals TX_CTL and RX_CTL. TX_EN and TX_ER are encoded in the TX_CTL signal as was described in the earlier section.

3.4 Station Management

The station management module is a simple communications core for reading and writing to PHY registers, it has a bidirectional MDIO signal, as well as a clock signal for the MDIO bus. Like most of the other cores in this report, its main component is a state machine and a number of counters. Figure 62 shows the main components of the station management entity.

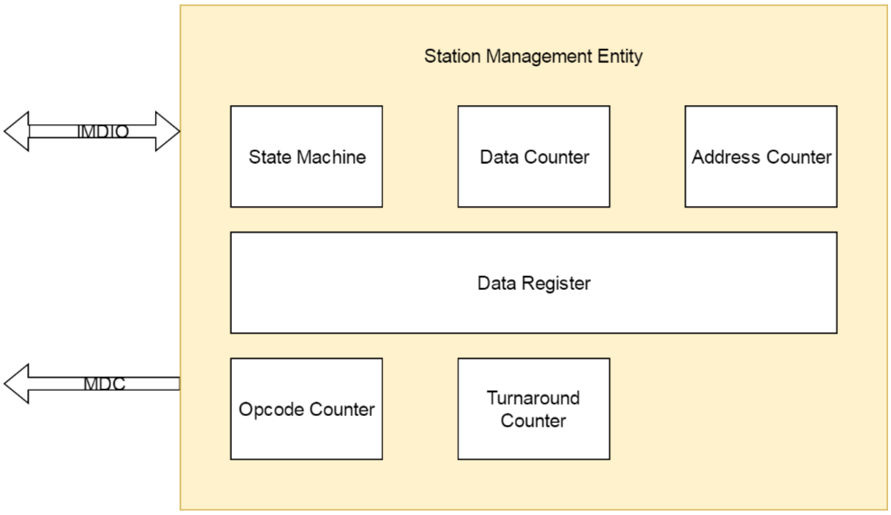


Figure 62 - Station Management

3.4.1 State Machine

The station management entity's purpose, along with the signalling structure is presented in section 2.2.3.5, this section will look at the design of the state machine that controls this signalling.

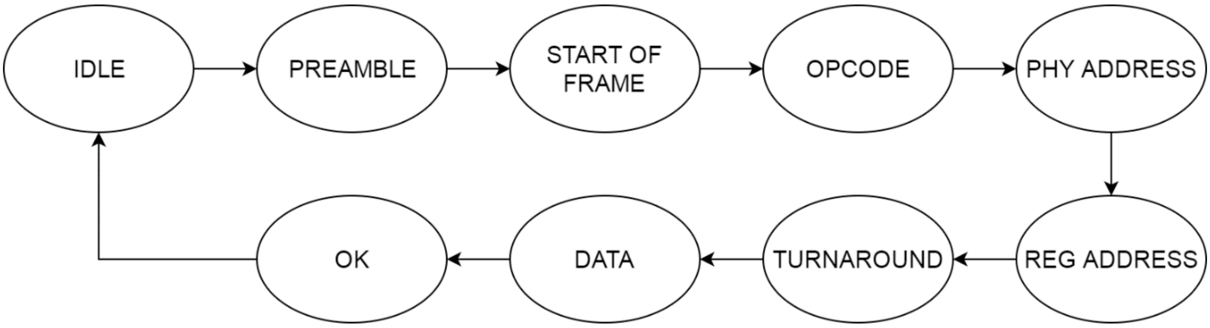


Figure 63 - Station Management - States

Figure 63 shows the states in the machine. All of the state transitions are as a result of a counter reaching zero, with the exception of the transition from the idle state, which is triggered by the begin transaction signal to the core.

In the preamble state 32 ones are transmitted onto the MDIO bus. The start of frame state is for transmission of the start of frame bits 0b10. The transmitted opcode is determined by the mode signal to the core, if this is a read transaction, then a 1 followed by a 0 is transmitted; if this is a write transaction then the reverse is sent. The 5-bit PHY address is sent next followed by the 5-bit register address. These are input to the core from the RS. In the turnaround state, a 1 followed by a 0 is transmitted onto the MDIO line for writing. If the transaction is a read transaction, then the core presents a high impedance on the MDIO line, allowing for the PHY to take control of the bus and the MDIO signal to become an input. In the data state, the 16-bits are either read from or written to the MDIO bus and a transition to the OK state follows this.

4 Implementation

4.1 Hardware

4.1.1 Electronics

The NetFPGA-1G-CML is a development board based on a Xilinx Kintex-7 FPGA. Its specification includes [20]:

- Xilinx Kintex-7 XC7K325T-1FFG676 FPGA
- Low-jitter 200 MHz oscillator
- Four 10/100/1000 Ethernet PHYs with RGMII.

Which will be used for the implementation of the MAC and to test the design in a real world situation.

Figure 64 - NetFPGA 1G CML [20]

The NetFPGA-1G-CML Platform is open hardware along with an open source code base [21] that together enable rapid prototyping of networking devices. The most important details of the hardware itself are the PCI Express Form Factor with 4 x 1 Gigabit Ethernet ports and a Xilinx Kintex-7 FPGA. The code base is based on the design flow of Xilinx Embedded Development Kit software and ARM's AXI-4 interface specification, the platform's open source code base includes reference projects, an IP library, and tools which assist in the development of new designs [20].

The NetFPGA-1G-CML Platform is documented on their GitHub wiki.

Whilst more expensive than some other boards at $1499, the NetFPGA-1G-CML has the advantage of much more support and example designs with respect to Ethernet. The

NetFPGA-1G-CML also has many peripherals such as DDR3, allowing for much more design flexibility [20].

4.2 Verilog

The following utilisation figures are estimates produced by Vivado for each of the modules post-synthesis. Vivado was configured for the NetFPGA and therefore a Kintex-7 XC7K325T-1FFG676 device. Packet counters were disabled for transmit and receive modules below. In a working test system, counters would not be needed as the counts could be inferred after the fact from the test data. Their impact is minimal, resulting in a small increase in flip-flops and LUTs

4.2.1 Receive

The estimated resource utilisation for the receive core can be seen in Table 5. Due to the counts that measure frame length, and the data register, which is used as a buffer between the input stream and the FIFO interface, the number of FFs is quite high relative to the number of LUTs. This is also due to the state machine logic, of which most of the resources are dedicated to.

4.2.1.1 Optimisation

In order to optimise the resource usage, the number of states are reduced to just the necessary ones. As more functionality is required, the number of states will undoubtedly increase, however it is estimated that the number of LUTs for this module should never exceed 200.

4.2.1.2 Utilisation

Table 5

Resource	Estimation	Available	Utilization %
LUT	123	203800	0.06
LUTRAM	8	64000	0.01
FF	100	407600	0.02

4.2.2 Transmit

The transmit module’s utilisation estimates are shown in Table 6. It is of similar design to the receive core, however, items such as the preamble, padding and FCS, all require a count to make sure the transmission of the respective portion of the packet is the correct length, resulting in a slightly increased usage over the receive module.

4.2.2.1 Optimisation

As with the receive core, the number of states has been kept to just the bare minimum.

4.2.2.2 Utilisation

Table 6

Resource	Estimation	Available	Utilization %
LUT	165	203800	0.08
LUTRAM	1	64000	0.01
FF	120	407600	0.03

4.2.3 CRC

The CRC core has been explicitly designed for low resource usage. In the following table it can be seen that the 32-bit CRC used in Ethernet has been implemented using only 12 LUTs and 5 FFs.

4.2.3.1 Optimisation

Use of a recursive algorithm to generate the core results in extremely low usage. One whitepaper proposing an algorithm for parallel CRC resulted in 161 LUTs and 32 FFs for a 32-bit parallel CRC.

4.2.3.2 Utilisation

Table 7

Resource	Estimation	Available	Utilization %
LUT	12	203800	0.01
FF	5	407600	0.01

4.2.4 Backoff

The backoff module is composed of a LFSR and two counters for the slot time and random count. Its utilisation can be seen in Table 8.

4.2.4.1 Utilisation

Table 8

Resource	Estimation	Available	Utilization %

LUT	30	203800	0.01
FF	29	407600	0.01

4.2.5 MII

Table 9 shows the usage for the MII module. A relatively high usage is seen due to the clock dividers for both transmit and receive, as well as the interface to the MAC and increasing the data width from 4 to 8 bits. This core will be optimised where possible in the future. As there was no immediate need for this module, the time was spent on optimising other cores.

4.2.5.1 Utilisation

Table 9

Resource	Estimation	Available	Utilization %
LUT	90	203800	0.04
FF	80	407600	0.02

4.2.6 RMII

With only one clock and therefore only one clock divider, the RMII module's usage is just over half of the MII's. The data path is only 2 bits wide and is brought up to 8-bits for the MAC interface, which adds slightly more usage. The data is available in Table 10.

4.2.6.1 Utilisation

Table 10

Resource	Estimation	Available	Utilization %
LUT	54	203800	0.03
FF	47	407600	0.01

4.2.7 GMII

4.2.7.1 Utilisation

The MAC interface is already GMII, therefore the RS module simply passes the signals through, requiring no extra logic be placed.

4.2.8 RGMII

The RGMII module for the most part is a pass-through module, as in the case with GMII. A small amount of logic is needed to encode and decode the RX_CTL and TX_CTL signals and this is seen in Table 11.

4.2.8.1 Utilisation

Table 11

Resource	Estimation	Available	Utilization %
LUT	4	203800	0.01

4.3 Xilinx Kintex-7 Implementation

The overall implementation consists of using the Xilinx Clocking Wizard to derive the transmit clock from the system clock and to buffer the receive clock. The Xilinx FIFO Generator is used to generate transmit and receive FIFOs (in loopback mode only one is needed). The DDR interfaces are taken care of by the Xilinx Select IO Wizard.

4.3.1 Constraints

A few constraints were put in place for the full implementation:

- Reset
    - Pins defined.
    - Set as false path.
        - Makes timing closure easier.
        - Input delay of reset button is negligible relative to human reaction time.
- System Clock.
    - Pins defined.
    - Period defined.
    - Jitter defined.
- RGMII Receive Clock.
    - Pin defined.
    - Period defined.
    - Jitter defined.
- RGMII Interface.

  - Pins defined.
  - Input delay defined.
- Transmit Clock.
  - Defined as a generated clock.
  - Period defined.
  - Jitter defined.

4.3.2 Timing Analysis

Timing analysis was completed after implementation. The margins are quite slim, and the longest timing critical paths by far include the CRC module. Its small footprint comes at the price of more difficult timing closure, a redesign of this core may be necessary in the future, however the FPGA this was implemented on was of the lowest speed grade that Xilinx offers. Implementation on a higher speed chip would be much easier. The tables below show the timing analysis summary from Vivado post-implementation.

Table 12

Setup	
Worst Negative Slack	0.037 ns
Total Negative Slack	0.000 ns
Number of Failing Endpoints	0
Total Number of Endpoints	501

Table 13

Hold	
Worst Hold Slack	0.088 ns
Total Hold Slack	0.000 ns
Number of Failing Endpoints	0
Total Number of Endpoints	501

Table 14

Pulse Width	
Worst Pulse Width Slack	1.100 ns
Total Pulse Width Slack	0.000 ns
Number of Failing Endpoints	0
Total Number of Endpoints	274

4.3.3 Total Utilisation

Table 15 shows the total utilisation for the entire Ethernet MAC, resource sharing allows the logic to be reduced across multiple cores, resulting in a total lower usage.

Table 15

Resource	Utilization	Available	Utilization %
LUT	360	203800	0.18
LUTRAM	2	64000	0.01
FF	248	407600	0.06
BRAM	0.50	445	0.11
IO	15	400	3.75
BUFG	4	32	12.50
MMCM	2	10	20.00

4.3.4 Area Utilisation

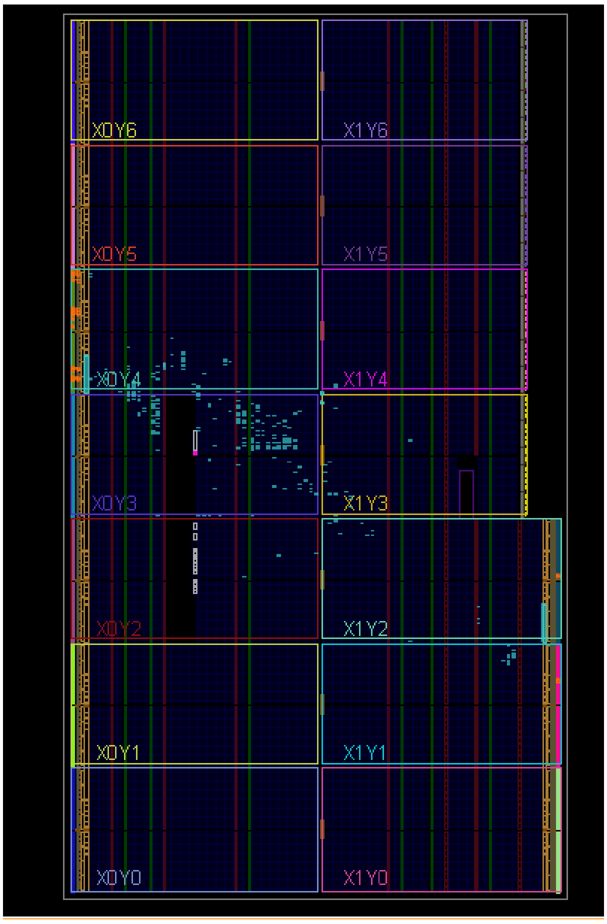


Figure 65 - On-Chip Area

As can be seen from Figure 65, the design is spread across a couple of slices. This was necessary during place-and-route to meet the timing constraints; ideally all of the logic would be grouped together. This is in part due to the differential system clock and reset signals being connected to pins that are at the opposite side of the FPGA to the RGMII signals, meaning that the path the clock has to take to reach all of the elements of the design is quite long.

4.3.5 Notes on Optimisation

This design has been optimised for resource usage, not timing. If this design is to be used as part of a larger design, then additional timing constraints would likely negatively impact the implementation. In the future, timing critical paths will need to be identified and minimised. As was previously mentioned, the CRC module is one area of concern with respect to timing.

4.3.6 Xilinx Kintex-7 Primitives

4.3.6.1 IDDR

The Xilinx 7-series devices have Input DDR (IDDR) registers in their I/O tiles. The IDDR primitive was instantiated by the Select I/O Wizard for the receive side of the RGMII signalling (RX_CTL and RXD). The receive clock is routed to the IDDR register and I/O delays are accounted for in the implementation stage in order to synchronise the clock and the data [22].

4.3.6.2 ODDR

There are also Output DDR (ODDR) registers available on the Kintex-7. This was instantiated in the same way that the IDDRs were for the design and are used for the transmit data and control signals [22].

4.3.6.3 BUFG

On defining a clock, a global buffer (BUFG) is instantiated to aid in the place-and-route stage. Global buffers are for distribution of signals with high fan-out, helping to reduce skew [22].

4.3.6.4 MMCM

The Mixed Mode Clock Manager (MMCM) is used to generate accurate clocks from input clocks with defined phase and frequency relationships. MMCMs are generated by the clocking wizard [23].

4.3.6.5 ILA

The Integrated Logic Analyser (ILA) is an IP core that creates a debug core on the JTAG chain inside the device. It can be used to monitor internal signals, and as such is invaluable when debugging a tricky design. The ILA was used during the testing stage in this project and its usage can be seen in the next section [24].

5 Testing

5.1 Verilog Test Benches

A number of Verilog test benches were written to simulate and test the design prior to implementation on a physical device. Designing in this way is much faster than working directly on the device from the very start, and allows for any major problems to be ironed out before attempting implementation.

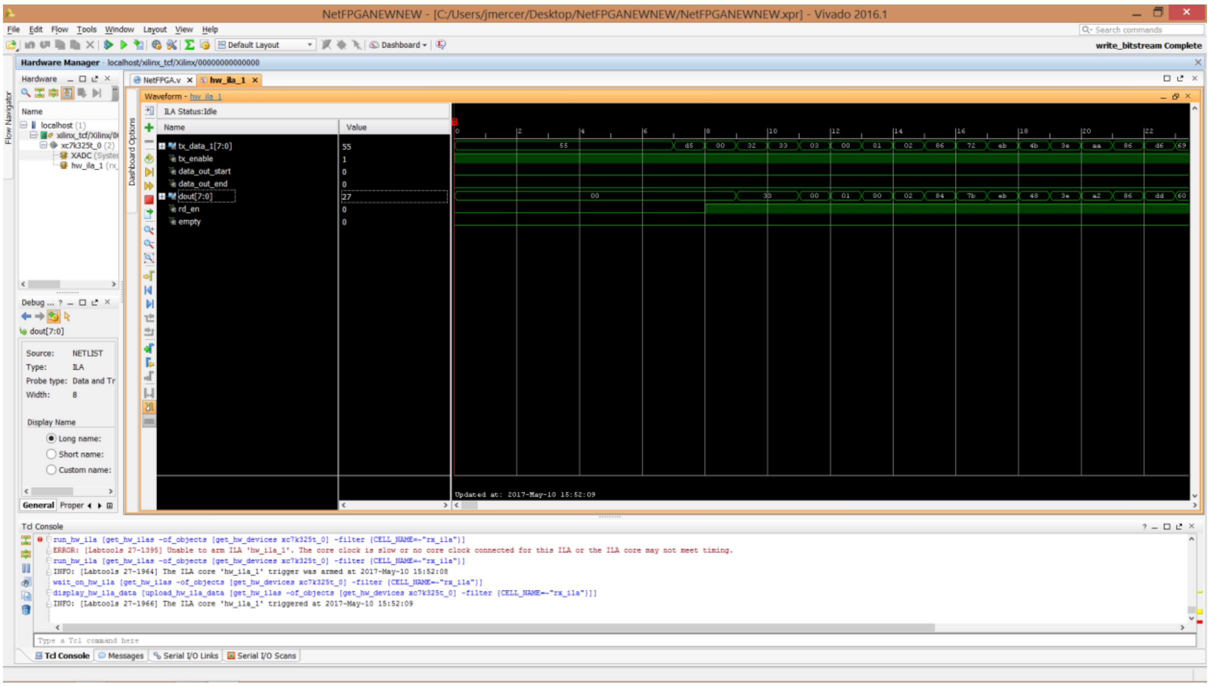
These test benches are available on the GitHub [19].

The test bench signal output was saved to a Value Change Dump (VCD) file and is displayed using GTKWave [25]. The wave outputs are too large to include in this report, but can be generated by running the test benches. Icarus Verilog [26] was used as a simulation tool.

The test benches are currently being written in an automated fashion so that they can be quickly used as a regression test whenever new features are added to the MAC. The automated test consists of a number of assertions that signals are of a particular value, and the number of passes and fails are recorded and presented to the user at the end of the test. Automated testing will become invaluable as the project grows.

5.2 Integrated Logic Analyser

Physical testing on the device is performed with the ILA, this was connected to over JTAG and the MAC was set up in loopback mode (any received data is looped back and transmitted by the transmit core). The NetFPGA was connected to a PC and the PC transmitted packets which needed to be received by the receive core and passed through a FIFO to the transmit side. Figure 66 shows the ILA in use from a debug core on the Kintex-7 [24].


NetFPGANEWNEW - [C:/Users/jmercer/Desktop/NetFPGANEWNEW/NetFPGANEWNEW.xpr] - Vivado 2016.1
Hardware Manager - localhost/xilinx_tcf/Xilinx/00000000000000
write_bitstream Complete
Waveform - hw_ila_1
ILA Status: Idle
tx_data_1[7:0]
tx_enable
data_out_start
data_out_end
dout[7:0]
rd_en
empty
Updated at: 2017-May-10 15:52:09
Tcl Console

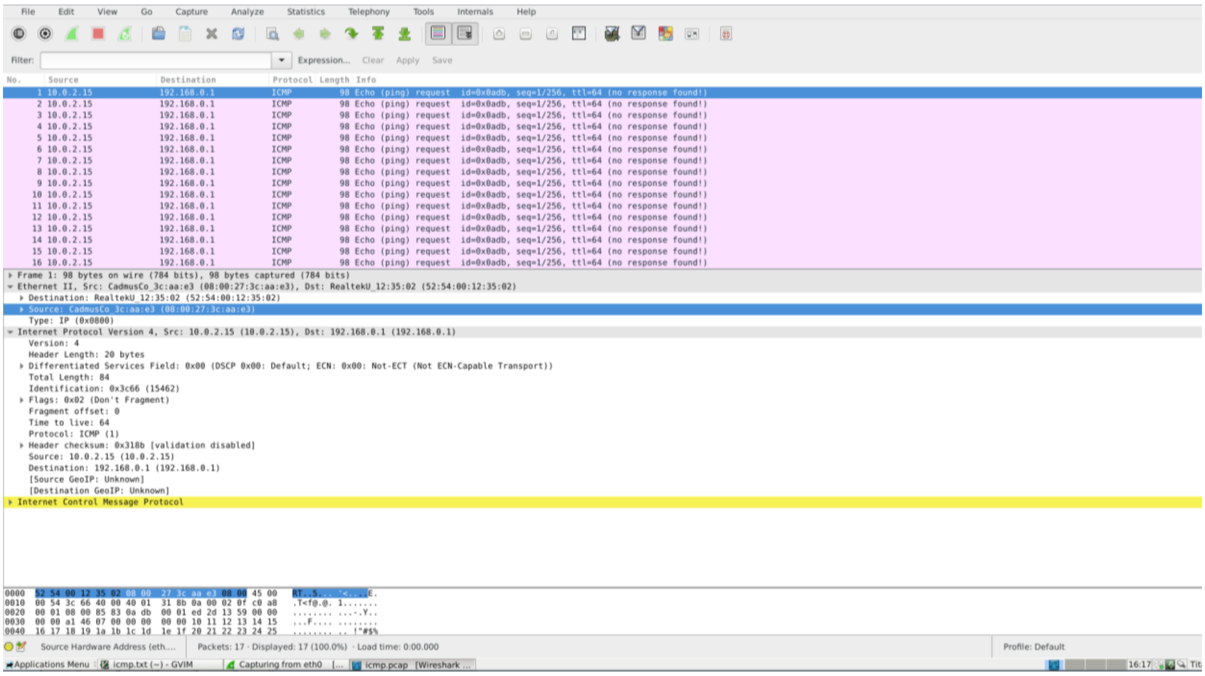
```
run_hw_ila [get_hw_ilas -of_objects [get_hw_devices xc7k325t_0] -filter {CELL_NAME=~"rx_ila"}]
ERROR: [Labtools 27-1395] Unable to arm ILA 'hw_ila_1'. The core clock is slow or no core clock connected for this ILA or the ILA core may not meet timing.
run_hw_ila [get_hw_ilas -of_objects [get_hw_devices xc7k325t_0] -filter {CELL_NAME=~"rx_ila"}]
INFO: [Labtools 27-1964] The ILA core 'hw_ila_1' trigger was armed at 2017-May-10 15:52:08
wait_on_hw_ila [get_hw_ilas -of_objects [get_hw_devices xc7k325t_0] -filter {CELL_NAME=~"rx_ila"}]
display_hw_ila_data [upload_hw_ila_data [get_hw_ilas -of_objects [get_hw_devices xc7k325t_0] -filter {CELL_NAME=~"rx_ila"}]]
INFO: [Labtools 27-1966] The ILA core 'hw_ila_1' triggered at 2017-May-10 15:52:09
```

Figure 66 - ILA

5.3 STC Packet Capture

The final way the implementation was tested was by using a packet capture to ensure that the MAC was operating correctly. The device was hooked up to the STC which was configured not to drop traffic; packets were sent and received on the STC and recorded so that the traffic could be inspected. Pings were sent from the STC at 100% line utilisation and were in turn received by the STC. Every packet that was sent was received, as expected in a loopback condition. The number of pings sent was recorded and the number of received packets was compared to this value and they were found to match. Other types of traffic was sent as well to give a range of sizes from 64 bytes to 1500 bytes. Each test was completed 3 times in order to give confidence in the results.

This set of tests proved that not only was the MAC operating correctly, but also that it could handle traffic at line rate with 64 byte packets.



```
No.   Source        Destination    Protocol Length Info
    1 10.0.2.15     192.168.0.1    ICMP         98 Echo (ping) request  id=0x0adb, seq=1/256, ttl=64 (no response found!)
    2 10.0.2.15     192.168.0.1    ICMP         98 Echo (ping) request  id=0x0adb, seq=1/256, ttl=64 (no response found!)
    3 10.0.2.15     192.168.0.1    ICMP         98 Echo (ping) request  id=0x0adb, seq=1/256, ttl=64 (no response found!)
    4 10.0.2.15     192.168.0.1    ICMP         98 Echo (ping) request  id=0x0adb, seq=1/256, ttl=64 (no response found!)
    5 10.0.2.15     192.168.0.1    ICMP         98 Echo (ping) request  id=0x0adb, seq=1/256, ttl=64 (no response found!)
    6 10.0.2.15     192.168.0.1    ICMP         98 Echo (ping) request  id=0x0adb, seq=1/256, ttl=64 (no response found!)
    7 10.0.2.15     192.168.0.1    ICMP         98 Echo (ping) request  id=0x0adb, seq=1/256, ttl=64 (no response found!)
    8 10.0.2.15     192.168.0.1    ICMP         98 Echo (ping) request  id=0x0adb, seq=1/256, ttl=64 (no response found!)
    9 10.0.2.15     192.168.0.1    ICMP         98 Echo (ping) request  id=0x0adb, seq=1/256, ttl=64 (no response found!)
   10 10.0.2.15     192.168.0.1    ICMP         98 Echo (ping) request  id=0x0adb, seq=1/256, ttl=64 (no response found!)
   11 10.0.2.15     192.168.0.1    ICMP         98 Echo (ping) request  id=0x0adb, seq=1/256, ttl=64 (no response found!)
   12 10.0.2.15     192.168.0.1    ICMP         98 Echo (ping) request  id=0x0adb, seq=1/256, ttl=64 (no response found!)
   13 10.0.2.15     192.168.0.1    ICMP         98 Echo (ping) request  id=0x0adb, seq=1/256, ttl=64 (no response found!)
   14 10.0.2.15     192.168.0.1    ICMP         98 Echo (ping) request  id=0x0adb, seq=1/256, ttl=64 (no response found!)
   15 10.0.2.15     192.168.0.1    ICMP         98 Echo (ping) request  id=0x0adb, seq=1/256, ttl=64 (no response found!)
   16 10.0.2.15     192.168.0.1    ICMP         98 Echo (ping) request  id=0x0adb, seq=1/256, ttl=64 (no response found!)
```

```
▸ Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
▾ Ethernet II, Src: CadmusCo_3c:aa:e3 (08:00:27:3c:aa:e3), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
  ▸ Destination: RealtekU_12:35:02 (52:54:00:12:35:02)
  ▸ Source: CadmusCo_3c:aa:e3 (08:00:27:3c:aa:e3)
    Type: IP (0x0800)
▾ Internet Protocol Version 4, Src: 10.0.2.15 (10.0.2.15), Dst: 192.168.0.1 (192.168.0.1)
    Version: 4
    Header Length: 20 bytes
  ▸ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 84
    Identification: 0x3c66 (15462)
  ▸ Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (1)
  ▸ Header checksum: 0x318b [validation disabled]
    Source: 10.0.2.15 (10.0.2.15)
    Destination: 192.168.0.1 (192.168.0.1)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
▸ Internet Control Message Protocol
```

```
0000  52 54 00 12 35 02 08 00  27 3c aa e3 08 00 45 00   RT..5... '<....E.
0010  00 54 3c 66 40 00 40 01  31 8b 0a 00 02 0f c0 a8   .T<f@.@. 1.......
0020  00 01 08 00 85 83 0a db  00 01 ed 2d 13 59 00 00   ........ ...-.Y..
0030  00 00 a1 46 07 00 00 00  00 00 10 11 12 13 14 15   ...F.... ........
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ........ .. !"#$%
```

Source Hardware Address (eth.... | Packets: 17 · Displayed: 17 (100.0%) · Load time: 0:00.000

Applications Menu | icmp.txt (~) - GVIM | Capturing from eth0 [... | icmp.pcap [Wireshark ...

Figure 67 - Packet Capture

6 Comparison with Existing IP

As can be seen from the following sections, the Ethernet MAC described in this report has vastly reduced resource utilisation than both the Xilinx MAC and the OpenCores MAC. These cores have additional features that are often not necessary, but even if they are ignored, the design outlined above has a very small footprint. The higher usage of the Xilinx MAC and the OpenCores MAC is partly due to the more pipelined nature of the designs which makes timing closure simpler.

6.1 Xilinx Tri-Mode Ethernet MAC

The Xilinx Tri-Mode Ethernet MAC (TEMAC) Core provides a Media Access Controller IP Block which can be used as a drop-in design element for Ethernet designs. It implements a 10/100/1000 Mb/s Ethernet MAC with several interface options. An overview of the TEMAC Core is shown below [27].

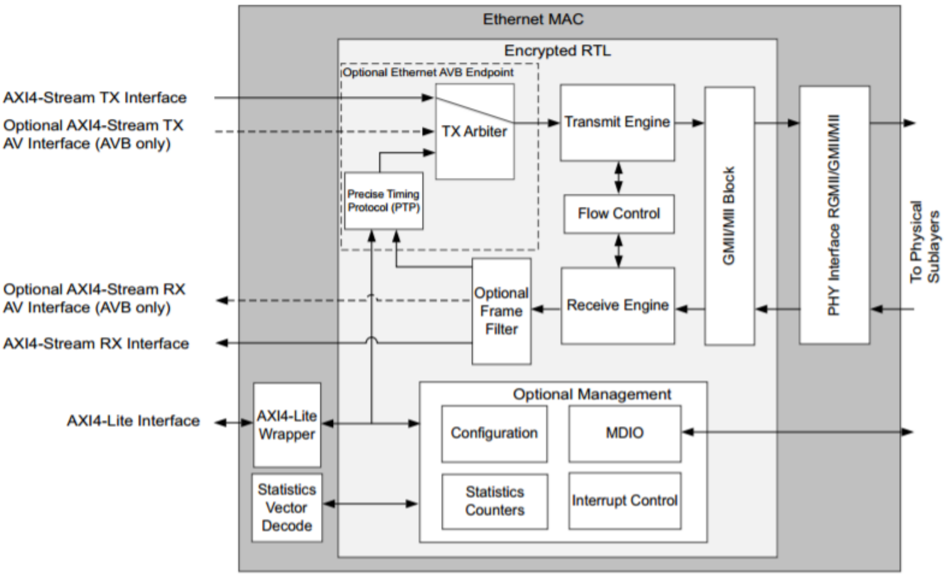


Figure 68 – Xilinx TEMAC Functional Block Diagram [27]

In the case of the NetFPGA-1G-CML the interface to the physical sublayers is Reduced Gigabit Media Independent Interface (RGMII) as detailed in the Background Information section.

The TEMAC Core connects to FPGA designs via a group of AXI4 interfaces:

- AXI4-Stream TX Interface
- AXI4-Stream Rx Interface

- AXI4-Lite Management Data I/O (MDIO)/Configuration Interface

A summary of the relevant features of this core include:

- Designed to the IEEE Std 802.3-2008 specification
- Supports separate IP cores
  - 10/100/1000 Mb/s Ethernet MAC
  - 1 Gb/s Ethernet MAC
  - 10/100 Mb/s Ethernet MAC
- Configurable duplex operation
- Support for MII, GMII, RGMII
- Optional Management Data Input/Output (MDIO) interface to manage objects in the physical layer
- User-accessible raw statistic vector outputs
- Optional built in statistics counters
- Support for VLAN frames
- Configurable IFG adjustment in full-duplex operation
- Configurable in-band FCS field passing on both transmit and receive paths
- Auto padding on transmit and stripping on receive paths
- Configurable support for jumbo frames of any length
- Configurable maximum frame length check
- AXI4-Stream user interface for Transmit and Receive frame data path.

6.1.1 Utilisation

Minimum possible utilisation for the TEMAC is outlined in Table 16, whereas the maximum is shown in Table 17 [28]. As can be seen, the resource utilisation is 4 to 5 times that of the Ethernet MAC in this report, with the exception of BRAMs which are not actually part of the MAC design, but are required for an actual implementation. The high utilisation on the TEMAC is most likely due to the AXI4-Stream interfaces and logic.

Table 16

LUTs	FFs	DSP48s	36k BRAMs	18k BRAMs
978	1526	0	0	0

Table 17

LUTs	FFs	DSP48s	36k BRAMs	18k BRAMs
1484	2516	0	0	0

6.2 OpenCores 10/100/1000 Ethernet MAC

The 10/100/1000 Mbps Ethernet MAC [29], as with the TEMAC provides MAC IP which can be dropped into a design. It was found that documentation for this was lacking, and it was quite difficult to synthesise the core due to code issues. This is in stark contrast to the TEMAC which has a plethora of information available for its use and implementation. This is probably due to the open-source nature of the project, and it has been poorly curated.

Its features include:

- Implements IEEE 802.3.
- Half-duplex support for 10 100 Mbps mode.
- FIFO interface.
- Flow control support.
- Source MAC address insertion.
- RMON MIB statistic counter

6.2.1 Utilisation

The utilisation for the OpenCores MAC is available in Table 18 [29]. Its utilisation is higher than the Xilinx core in terms of LUTs but with reduced FF usage, opting to use BRAMs instead. The Ethernet MAC in this report has much lower usage on all fronts.

Table 18

LUTs	FFs	DSP48s	36k BRAMs	18k BRAMs
1526	1198	0	0	4

7 Future Development

In addition to the comments made about improvements to this core throughout the document, there are a few additional future development opportunities.

7.1 IEEE 802.2 Logical Link Control Core

Logical Link Control (LLC) sits directly above the MAC sublayer and performs tasks such as Automatic Repeat Request (ARQ) and flow control. It would be a useful addition when testing if LLC functionality is a required part of the test, and could be enabled on a test by test basis. An LLC core is a future task for this project.

7.2 VLAN Support with Quality of Service

More often businesses are using Quality of Service (QoS) to police and shape traffic. A welcome addition to this core would be the ability to push, pop, and rewrite VLAN tags as well as recognise different levels of QoS. QoS features heavily in Asidua's product offering.

7.3 Packet Generator & Shaper

In order to test traffic through a DUT, packets must be sent. In its current state, the MAC can be set up to repeatedly send the same packet, however, in a real test environment it would be useful to be able to send several streams of packets to simulate real-world behaviour. Shaping is the scheduling and rate limiting of packet streams; DUTs can often perform rate-limiting, and therefore a useful metric is being able to send packet streams at a specific rate.

7.4 RFC 2544

Request for Comments (RFC) 2544 was a document produced by the Internet Engineering Task Force (IETF) in 1999 which discusses and defines several tests which can be used to assess the performance of a network interconnecting device. RFC 2544 was designed to eliminate confusion for consumers when purchasing a network device; vendors in the past would often use a "smoke and mirrors" approach to their marketing, reporting various product characteristics which don't provide a full picture of a network device that could be used to compare it with other devices [30]. An example of this sort of practice can be seen with Volkswagen's recent emissions testing debacle [31].

RFC 2544 defines a test setup which can be seen below. The two options are a single testing unit, or a centrally controlled, but separate, sender and receiver.

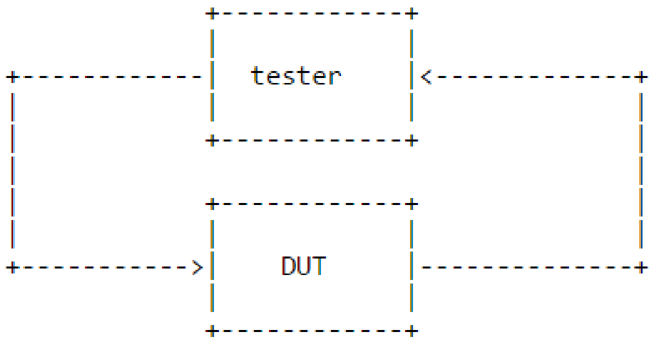


Figure 69 – RFC 2544 Test Setup 1

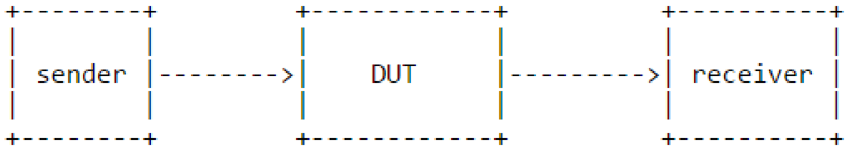


Figure 70 – RFC 2544 Test Setup 2 [30]

The Device under Test (DUT) can be a single device or a network of devices in this context, allowing for simultaneous testing of two devices, across multiple media.

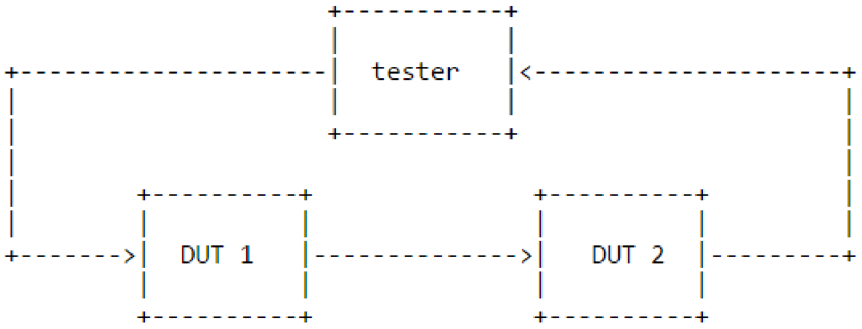


Figure 71 – RFC 2544 Testing of Multiple Devices [30]

The standard also defines test frame formats and sizes to give a view of a devices performance across a range of use cases. For example, a device may not be able to handle a full rate of minimum sized 64 byte IP packets without dropping frames, due to the increased overhead, but may be able to handle 128 byte packets without issue.

RFC 2544 provides engineers and network technicians with a common language and results format.

The RFC 2544 describes six subtests:

- Throughput: Measures the maximum rate at which none of the frames are dropped by the device/system under test (DUT/SUT).
- Back-to-back or burstability: Measures the longest burst of frames at maximum throughput or minimum separation between frames that the DUT/SUT will handle without any loss of frames. This measurement is a good indication of the buffering capacity of a DUT.
- Frame loss: Defines the portion of frames that should have been forwarded by a network device under a steady state load that were not forwarded. This measurement can be used for reporting the performance of a network device in an overloaded state, as it can be a useful indication of how a device would perform under network conditions such as broadcast storms.
- Latency: Measures the Round-Trip Time (RTT) taken by a test frame to travel through a network device or across the network and back to the test port. Latency is the time interval that begins when the last bit of the input frame reaches the input port and ends when the first bit of the output frame is seen on the output port. It is the time taken by a bit to go through the network and back. Latency variability can be a problem. With protocols like voice over Internet protocol (VoIP), a variable or long latency can cause degradation in voice quality.
- System reset: Measures the speed at which a DUT recovers from a hardware or software reset. This subtest is performed by measuring the interruption of a continuous stream of frames during the reset process.
- System recovery: Measures the speed at which a DUT recovers from an overload condition. This subtest is performed by temporarily overloading the DUT and then reducing the throughput at normal or low load while measuring frame delay in these two conditions.

Conformance to RFC2544 is a target for all network testing devices.

7.5 ITU-T Y.1564

ITU-T Y.1564 is a standard that defines a test methodology that allows for complete validation of Ethernet Service Level Agreements (SLA) in a single test [32]. A Service Level Agreement is an agreement between a service provider and a customer which sets out service commitments such as Quality, Availability, Network Boundaries, Fault Reporting, Bandwidth, Jitter, etc.

A tool conforming to ITU-T Y.1564 can be used to verify that a connection conforms to Performance Metrics set out in the customer’s SLA.

While RFC 2544 provides a methodology for benchmarking a DUT, it is designed to be used in a laboratory setting; ITU-T Y.1564 is designed as a more up-to-date in place testing methodology allowing for benchmarking to be completed outside of a lab setting. The context of an out-of-lab system is shown below, with User Network Interfaces (UNI) labelled. The metrics that can be measured are inherited from ITU-T Y.1563.

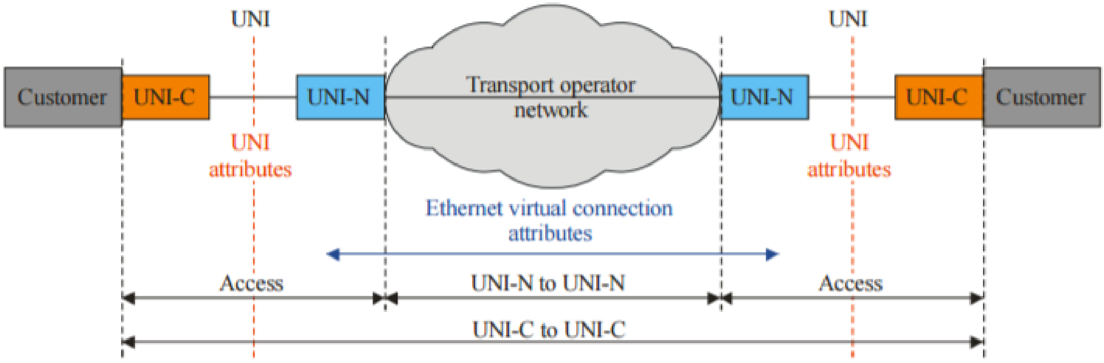


Figure 72 – ITU-T Y.1564 Definition of a Customer Connection across a Transport Network [32]

7.6 EOAM Core

Ethernet Operation, Administration, and Maintenance (EOAM) is another technology that feature heavily in Asidua's products and as such, needs to be tested. A dedicated core for this task would make a great addition to the testing unit.

8 Project Management

8.1 Overall Project Timeline

Date	Notes
3rd October 2016	Project started & initial specification discussed
12th October 2016	NetFPGA-1G-CML board selected and ordered
17th October 2016	NetFPGA Development Board arrived & initial clone of NetFPGA repository made.
14th November 2016	Project delayed up until this point due to licensing issues. PC set up with Linux and Environment setup begins.
5th December 2016	NetFPGA Reference NIC synthesis working after editing of Makefiles and experimentation with software versions of Vivado, ISE and supporting software.
12th December 2016	NetFPGA repository causing issues with conflicting versions of software across projects and out-of-date Makefiles.
19th December 2016	Attempting and failing to get simulations working with the NetFPGA repository.
27th December 2016	Beginning a review of the development process so far. Issues with NetFPGA repository are noted.
2nd January 2017	Decision made to move away from the NetFPGA repository and use it as reference only. Research into newer Vivado tools such as Board Interfaces started.
19th January 2017	Planning for the next stage of the project undertaken.
1st February 2017	Xilinx TEMAC suitability for the project evaluated.
10th February 2017	OpenCores Ethernet MAC simulations running.
14th February 2017	Decided that the TEMAC license is too expensive. OpenCores MAC synthesizing.
20th February 2017	OpenCores MAC not passing timing. The possibility of designing a MAC that is more suitable for the project is considered.
27th February 2017	Decision taken to change the scope and design an Ethernet MAC which fits the project better.

8.2 Development Timeline

Date	Notes
1st March 2017	MAC design started.
5th March 2017	Rudimentary design completed. Verilog coding begins.
9th March 2017	Backoff module development is complete and the core is simulated and passing the tests.
20th March 2017	CRC module is complete and simulated.
27th March 2017	Transmit module is complete and simulated.
5th April 2017	Receive module is complete and simulated.
7th April 2017	MII module is complete and simulated.
8th April 2017	RMII module is complete and simulated.
9th April 2017	GMII module is complete and simulated.
10th April 2017	RGMII module is complete and simulated.
14th April 2017	Constraints set up for Kintex-7.
20th April 2017	Design connected with Xilinx IP instantiated.
24th April 2017	Tested with ILA.
25th April 2017	Loopback testing in DS lab started.
3rd May 2017	Project expo.

8.3 Risks & Mitigation

The project as a whole has faced many issues, ultimately resulting in the scope for this report being defined as designing an Ethernet MAC for a network testing device. This was considered to be a core component of the overarching project and could not be overlooked.

At the outset, the project was intended to use and extend the Open Source Network Tester [33] (OSNT) which is a part of the NetFPGA project based on the 10G version of the NetFPGA boards. Although it is open source, it uses the closed source Xilinx TEMAC for connection to the PHYs on the 10G board. On evaluation of this MAC, the extensions that were initially planned would not have been possible. Now that this open source MAC has been developed, the rest of the project can go ahead, and the OSNT project can be ported to the NetFPGA 1G board, and the Xilinx IP removed.

Thousands of developers have contributed to the NetFPGA project, and what was once a well organised and defined design flow, and repository, has become a hard to maintain system and a convoluted process for an end user or new developer. This is compounded by the fact that the Xilinx tools have moved on by many versions and the prerequisite versions of Xilinx Vivado, Xilinx EDK and Xilinx ISE, along with other software prerequisites, are very specific to each project and is not homogenous across the entire repository. This is something which has been experienced by many new developers on the NetFPGA project, especially those who are not students at either Stanford or Cambridge, and cannot easily get assistance from the maintainers of the project.

For a lot of projects the build system is broken and out-of-date, with projects being built for the wrong FPGA device, requiring conflicting versions of Xilinx tools and support software, and a general disorganisation within projects. The repository is vast and encompasses many projects and cores making it hard to navigate. Items that should be centralised, such as board-related constraints (pins, etc.) are not, such that new EDK projects have to redefine these things.

Building a code base for the NetFPGA from the ground up, as in the case with this project is an opportunity to reorganise the repository and make it user-friendly. After the MAC and the tasks outlined in section 7 have been completed, the NetFPGA OSNT project will be ported, module by module, into the new repository.

8.4 Gantt Chart

Gantt charts for the MAC development and for the project as a whole are available in the appendices.

9 Conclusion

Over the course of the last few months we have set out to produce a network testing device. This is no small task and as such, has needed to be broken up into smaller, more manageable chunks. The first major milestone of the overarching project has been reached with the device now having the core component of being able to communicate over Ethernet using the Ethernet MAC described in this report.

We have looked at the operation of Ethernet and its ins and outs with respect to the MAC layer. Functionality from CSMA/CD to packet bursting and carrier extension have been explained and the most relevant features rolled into the design of the Ethernet MAC.

The MAC was designed with simplicity of internal operation and low resource usage in mind, and I would say that those goals have been met. The MAC can now just be dropped into a design and used, and while there are a few updates planned over the next few weeks, it is completely usable as it stands.

The Ethernet MAC in this report supports operation at 10, 100, and 1000 Mbps across the 4 most common MIIs at those speeds. Generalising the MAC for operation at 10Gbps and beyond is part of a longer term plan for the Ethernet testing device and will not become an essential feature for a long time.

Prior implementations of Ethernet MACs have been shown to have much higher resource usage than the one outlined in this report with LUT and FF usage in this design less than a quarter of the Xilinx and OpenCores designs.

All of the major features of the MAC sublayer were included. The MAC supports operation at both half duplex and full duplex, and can receive and drop packets with errors in length, FCS, or that have an error inserted via the RX_ER signal.

Testing was completed first through simulations using Icarus Verilog, prototypes of automated test benches have been produced and the design of these will need to be consolidated before moving on with the project; regression testing is an important tool.

The project has been implemented on a Xilinx Kintex-7 present on the NetFPGA 1G CML development board, and has now become part of the much larger open source NetFPGA project, providing the option for NetFPGA developers to move away from using the Xilinx TEMAC and use the open source and free Ethernet MAC described in this report.

From an overall project standpoint, the MAC was the first step towards production of a working traffic testing system. Now that the MAC is in a working state, progress on the other parts of the network testing unit can be completed by working through the list of features in the future developments section of this report. A long term project plan has been produced and will be started in earnest in summer 2017.

10 Bibliography

[1] Texas Instruments, “CAN bus, Ethernet, or FPD-Link: Which is best for automotive communications?,” 2014. [Online]. Available: http://www.ti.com/lit/an/slyt560/slyt560.pdf.

[2] RapidIO Trade Association, “RapidIO, PCI Express and Gigabit Ethernet Comparison,” 03 May 2005. [Online]. Available: http://www.rapidio.org/files/InterconnectComparison_v02.pdf.

[3] IEEE, “200 Gb/s and 400 Gb/s Ethernet Task Force,” [Online]. Available: http://www.ieee802.org/3/bs/.

[4] J. Z. Charles E. Spurgeon, Ethernet The Definitive Guide, O'Reilly, 2014.

[5] P. M. Rasmussen, “Comparison of twisted pair based ethernet technologies,” 2016.

[6] Huawei, “Thoughts on the practicality of Terabit Ethernet,” [Online]. Available: http://www.ieee802.org/3/ad_hoc/hse/public/12_09/cui_hse_01a_0912.pdf.

[7] IETF, “A Standard for the Transmission of IP Datagrams on Avian Carriers,” 01 04 1990. [Online]. Available: https://tools.ietf.org/html/rfc1149.

[8] IETF, “Scenic Routing for IPv6,” 01 04 2015. [Online]. Available: https://tools.ietf.org/html/rfc7511.

[9] IEEE, IEEE 802.3 Ethernet, 2015.

[10] InductiveLoad, “Diagram showing the structure of a MAC-48 network address, explicitly showing the positions of the multicast/unicast bit and the OUI/local address type bit.,” 2007.

[11] RMII Consortium, “RMII Specification,” 20 March 1998. [Online]. Available: http://ebook.pldworld.com/_eBook/-Telecommunications,Networks-/TCPIP/RMII/rmii_rev12.pdf.

[12 HP, “Reduced Gigabit Media Independent Interface,” 2002. [Online]. Available:
] http://web.archive.org/web/20160303171328/http://www.hp.com/rnd/pdfs/RGMIIv2_0_final_hp.pdf.

[13 Fcorthay, “MIIM write and read access,” [Online]. Available:
] https://en.wikipedia.org/wiki/Management_Data_Input/Output#/media/File:Miim_timing.svg.

[14 Xilinx, “7 Series FPGAs Configurable Logic Block,” [Online]. Available:
] https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf.

[15 Altera, “White Paper - FPGA Architecture,” [Online]. Available:
] https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/wp/wp-01003.pdf.

[16 InductiveLoad, “D Flip Flop Symbol,” [Online]. Available:
] https://en.wikipedia.org/wiki/Flip-flop_(electronics)#/media/File:D-Type_Flip-flop.svg.

[17 Cburnett, “2 to 1 mux,” [Online]. Available:
] https://en.wikipedia.org/wiki/Multiplexer#/media/File:Multiplexer_2-to-1.svg.

[18 D. I. K. Gorontzi, “Online CRC Calculation,” [Online]. Available:
] https://www.ghsi.de/CRC/indexDetails.php?Polynom=100101&Message=36.

[19 GitHub, “GitHub - JakeMercer,” [Online]. Available: https://github.com/JakeMercer.
]

[20 Digilent, “NetFPGA-1G-CML Kintex-7 FPGA Development Board,” [Online]. Available:
] http://store.digilentinc.com/netfpga-1g-cml-kintex-7-fpga-development-board/.

[21 GitHub, “NetFPGA 1G CML Live,” [Online]. Available:
] https://github.com/NetFPGA/NetFPGA-1G-CML-live.

[22 Xilinx, “7 Series FPGAs SelectIO Resources,” [Online]. Available:
] https://www.xilinx.com/support/documentation/user_guides/ug471_7Series_SelectIO.pdf.

[23] Xilinx, “7 Series FPGAs Clock Resources,” [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug472_7Series_Clocking.pdf.

[24] Xilinx, “Integrated Logic Analyzer v6.0,” [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/ila/v6_1/pg172-ila.pdf.

[25] GTKWave, “GTKWave,” [Online]. Available: http://gtkwave.sourceforge.net/.

[26] Icarus, “Icarus Verilog,” [Online]. Available: http://iverilog.icarus.com/.

[27] Xilinx, “LogiCore IP Tri-Mode Ethernet MAC v8.2,” [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/tri_mode_ethernet_mac/v8_2/pg051-tri-mode-eth-mac.pdf.

[28] Xilinx, “Resource Utilization for Tri Mode Ethernet MAC v9.0,” [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/ru/tri-mode-ethernet-mac.html.

[29] OpenCores, “10_100_1000 Mbps tri-mode ethernet MAC :: Overview,” [Online]. Available: http://opencores.org/project,ethernet_tri_mode.

[30] IETF, “Benchmarking Methodology for Network Interconnect Devices,” March 1999. [Online]. Available: https://tools.ietf.org/html/rfc2544.

[31] BBC, “Volkswagen: The scandal explained,” [Online]. Available: http://www.bbc.co.uk/news/business-34324772.

[32] ITU-T, “Y.1564 : Ethernet service activation test methodology,” [Online]. Available: https://www.itu.int/rec/T-REC-Y.1564/en.

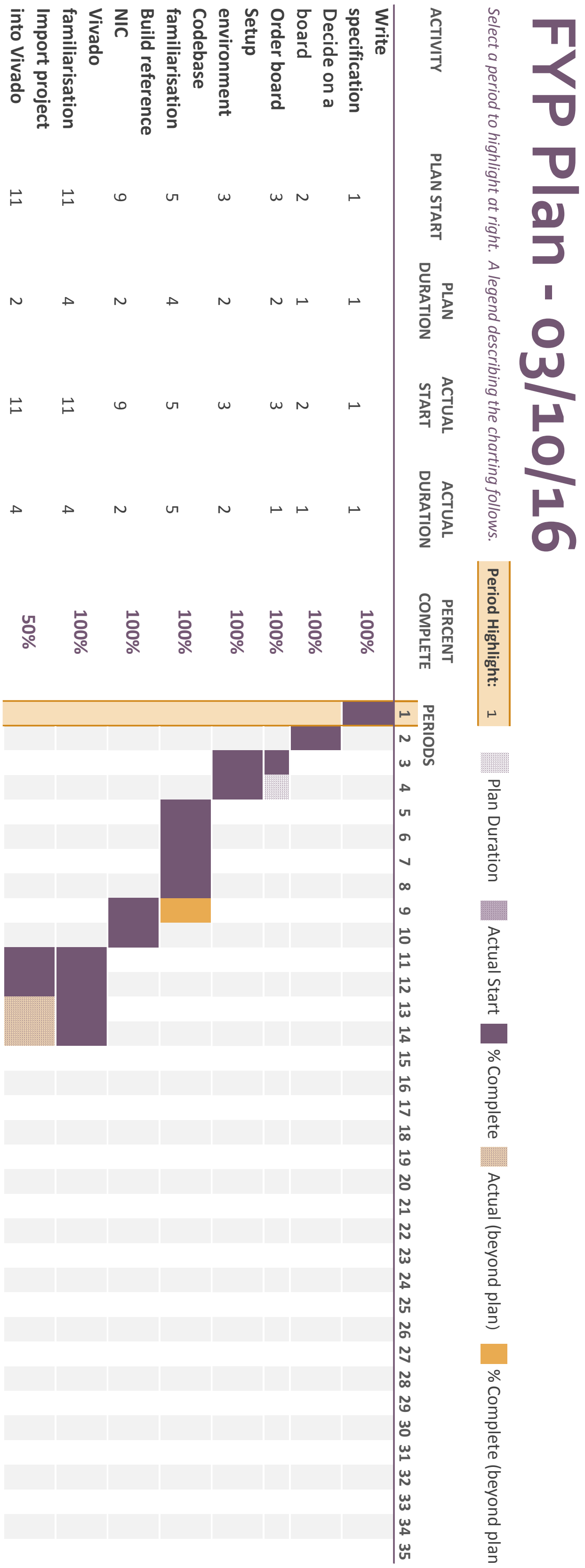
[33] OSNT, “OSNT,” [Online]. Available: http://osnt.org/.

11 Appendices

11.1 Appendix A – CRC Code Snippet

```
function automatic prev;
   input integer level;
   input integer index;
   input [DATA_WIDTH-1:0]  data;
   input [CRC_WIDTH-1:0]   crc;
   begin
      if (POLYNOMIAL[index])
         if (level)
            if (index)
                  prev = prev(level-1, (index)?index-1:CRC_WIDTH-1, data, crc) ^ prev(level-1, 
CRC_WIDTH-1, data, crc) ^ data[DATA_WIDTH-1-level];
            else
               prev = prev(level-1, CRC_WIDTH-1, data, crc) ^ data[DATA_WIDTH-1-level];
         else
            if (index)
               prev = crc[index-1-:1] ^ crc[CRC_WIDTH-1-:1] ^ data[DATA_WIDTH-1-level];
            else
               prev = crc[CRC_WIDTH-1-:1] ^ data[DATA_WIDTH-1-level];
      else
         if (level)
            if (index)
               prev = prev(level-1, index-1, data, crc);
            else
               prev = prev(level-1, CRC_WIDTH-1, data, crc) ^ data[DATA_WIDTH-1-level];
         else
            if (index)
               prev = crc[index-1-:1];
            else
               prev = crc[CRC_WIDTH-1-:1] ^ data[DATA_WIDTH-1-level];
   end
endfunction
```

11.2 Appendix B – Gantt Chart – October



FYP Plan - 03/10/16

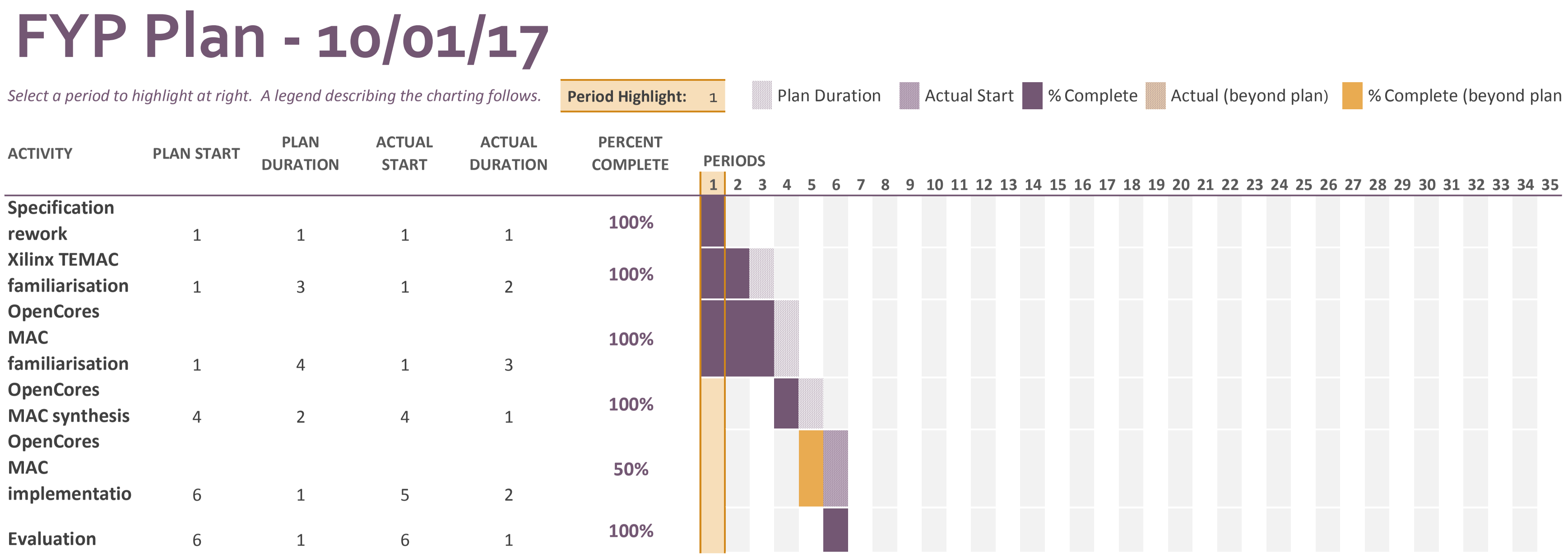
Select a period to highlight at right. A legend describing the charting follows.

Period Highlight: 1

▨ Plan Duration ▨ Actual Start ■ % Complete ▨ Actual (beyond plan) ■ % Complete (beyond plan

ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
Write specification	1	1	1	1	100%
Decide on a board	2	1	2	1	100%
Order board	3	2	3	1	100%
Setup environment	3	2	3	2	100%
Codebase familiarisation	5	4	5	5	100%
Build reference NIC	9	2	9	2	100%
Vivado familiarisation	11	4	11	4	100%
Import project into Vivado	11	2	11	4	50%

11.3 Appendix C – Gantt Chart – January



FYP Plan - 10/01/17

Select a period to highlight at right. A legend describing the charting follows.

Period Highlight: 1

ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
Specification rework	1	1	1	1	100%
Xilinx TEMAC familiarisation	1	3	1	2	100%
OpenCores MAC familiarisation	1	4	1	3	100%
OpenCores MAC synthesis	4	2	4	1	100%
OpenCores MAC implementatio	6	1	5	2	50%
Evaluation	6	1	6	1	100%

11.4 Appendix D – Gantt Chart – March

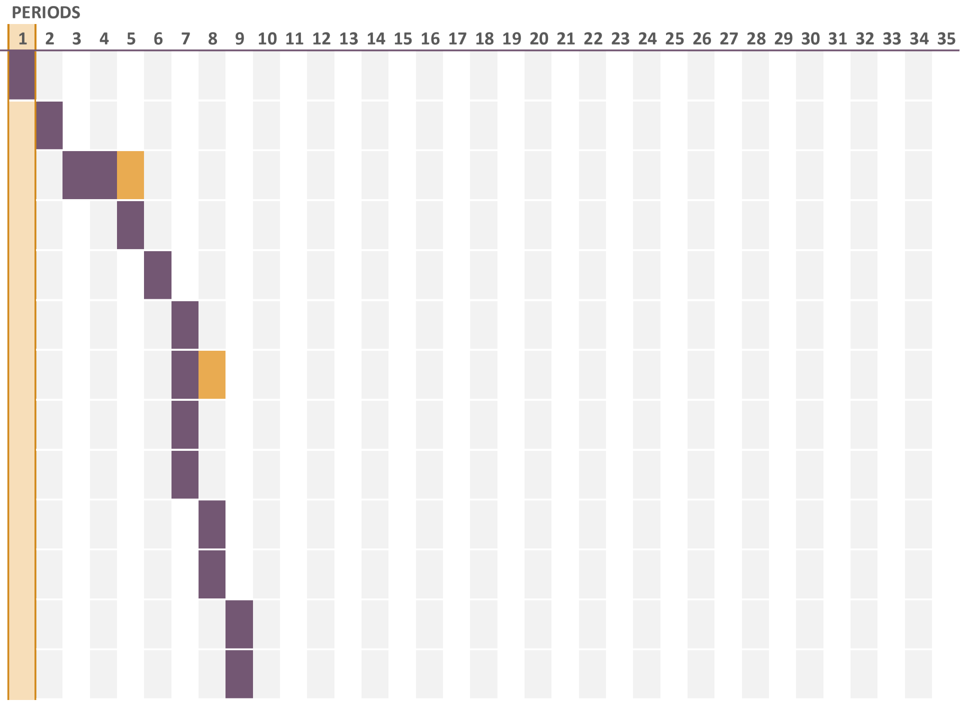
FYP Plan - 01/03/17

Select a period to highlight at right. A legend describing the charting follows.

Period Highlight: 1

▨ Plan Duration ▨ Actual Start ■ % Complete ▨ Actual (beyond plan) ■ % Complete (beyond plan

ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
MAC Design	1	1	1	1	100%
Backoff Module	2	1	2	1	100%
CRC Module	3	2	3	3	100%
Transmit Module	5	1	5	1	100%
Receive Module	6	1	6	1	100%
MII Module	7	1	7	1	100%
RMII Module	7	1	7	2	100%
GMII Module	7	1	7	1	100%
RGMII Module	7	1	7	1	100%
Constraints	8	1	8	1	100%
Implementation	8	1	8	1	100%
On-Chip Testing	9	1	9	1	100%
Testing with STC	9	1	9	1	100%


PERIODS