An extended abstract on

ETHERNET

MEDIA ACCESS CONTROL PROTOCOL

(EMAC)

Designing in Bluespec

Submitted by

CHINNAM VISHAL KUMAR

(EE11B012)

Under the guidance of

Prof. KAMAKOTI V

In partial fulfillment of the requirement for
The award of the degree of

DUAL DEGREE

In

ELECTRICAL ENGINEERING

VLSI SPECIALIZATION

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS
CHENNAI 600036

THESIS CERTIFICATE

This is to certify that the thesis entitled "**ETHERNET MEDIA ACCESS CONTROLLER DESIGNING IN BLUESPEC"** submitted by **CHINNAM VISHAL KUMAR** to the Indian Institute of Technology Madras, for the award of the Degree of Master in VLSI and Bachelor of Technology in Electrical Engineering in the Department of Electrical Engineering is a **bona fide record of project work** carried out by him under our supervision. The content of this thesis, in full or in part, have not been submitted to any other Institution or University for the award of any degree or diploma.

Prof. Kamakoti V
Professor
Department of Computer Science and Eng.
Indian Institute of Technology Madras

Prof. Harishankar Ramachandran
Head of Department
Department of Electrical Engineering
Indian Institute of Technology Madras

Prof. Nitin Chandrachoodan (co-guide)
Professor
Department of Electrical Engineering
Indian Institute of Technology Madras

Place: Chennai

Date:

Signature of the guide:

ACKNOWLEDGEMENTS

It gives me great pleasure in expressing my gratitude and a sincere thanks to my project guide, Prof. Kamakoti, who gave me this great topic of my interest as my final year project. I would like to thank the MS and PhD students under him who also guided me throughout the term of this work, giving valuable suggestions and direction at necessary moments which was crucial in the timely completion of present work. I would like to convey my deep regards to them for their patience, careful attention to details and constant motivation which helped me immensely in completing this project. I would also like to thank my co-guide Prof. Nitin Chandrachoodan for the support given.

I express my utmost regards to Head of Department, Electrical Engineering and all other professors of Department of Electrical Engineering for their constant support and feedback throughout the course of my degree.

I also express my gratitude to my Guru and my brothers who have guided me in my life giving me constant support to make me grow in the right path. I would like to dedicate my work to them.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Bluespec SystemVerilog (BSV) is aimed at hardware designers who are using or expect to use Verilog [IEE05], VHDL [IEE02], SystemVerilog [IEE13], or SystemC [IEE12] to design ASICs or FPGAs. It is also aimed at people creating synthesizable models, transactors, and verification components to run on FPGA emulation platforms. BSV substantially extends the design subset of SystemVerilog, including SystemVerilog types, modules, module instantiation, interfaces, interface instantiation, parameterization, static elaboration, and "generate" elaboration. BSV can significantly improve the hardware designer's productivity with some key innovations:

• It expresses synthesizable behavior with Rules instead of synchronous always blocks. Rules are powerful concepts for achieving correct concurrency and eliminating race conditions. Each rule can be viewed as a declarative assertion expressing a potential atomic state transition. Although rules are expressed in a modular fashion, a rule may span multiple modules, i.e., it can test and affect the state in multiple modules. Rules need not be disjoint, i.e., two rules can read and write common state elements. The BSV compiler produces efficient RTL code that manages all the potential interactions between rules by inserting appropriate arbitration and scheduling logic, logic that would otherwise have to be designed and coded manually. The atomicity of rules gives a scalable way to avoid unwanted concurrency (races) in large designs.

• It enables more powerful generate-like elaboration. This is made possible because in BSV, actions, rules, modules, interfaces and functions are all first-class objects. BSV also has more general type parameterization (polymorphism). These enable the designer to "compute with design fragments," i.e., to reuse designs and to glue them together in much more flexible ways. This leads to much greater succinctness and correctness.

• It provides formal semantics, enabling formal verification and formal design-by-refinement. BSV rules are based on Term Rewriting Systems, a clean formalism supported by decades of theoretical research in the computer science community [Ter03]. This, together with a judicious choice of a design subset of SystemVerilog, makes programs in BSV amenable to formal reasoning.

CHAPTER 1

INTRODUCTION

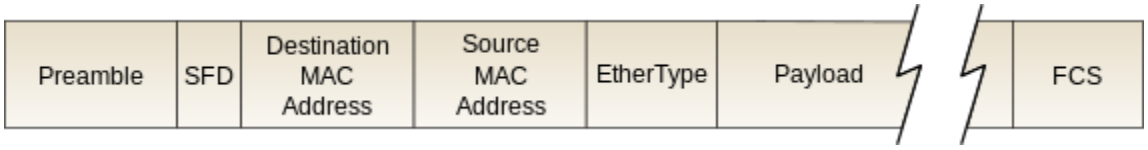
1.1 Project description

The project is about implementing the Ethernet MAC protocol in Bluespec. The MAC protocol is used to send and receive data packets in a medium (either wired or wireless). This project implements the way the packets are sent by a transmitter and received by a receiver according to the MAC protocol.

Ethernet MAC Protocol Overview

The Ethernet provides an unreliable, connectionless service to a networking application. A brief overview of the Ethernet protocol follows.

- All the Ethernet technologies use the same frame structure.
- The format of an Ethernet frame is shown below. The Ethernet packet is the collection of bytes representing the data portion of a single Ethernet frame on the wire.
- The Ethernet frames are of variable lengths, with no frame smaller than 64 bytes or larger than 1518 bytes (excluding preamble and SFD).
- An Ethernet frame is preceded by a preamble and start frame delimiter (SFD), which are both part of the Ethernet packet at the physical layer.
- Each Ethernet frame starts with an Ethernet header, which contains destination and source MAC addresses as its first two fields.
- The middle section of the frame is payload data including any headers for other protocols (for example, Internet Protocol) carried in the frame.
- The frame ends with a frame check sequence (FCS), which is a 32-bit cyclic redundancy check used to detect any in-transit corruption of data
- Figure 1: An Ethernet frame inside an Ethernet packet, with SFD marking the end of the packet preamble and indicating the beginning of the frame



Field	Bytes	Description
Preamble	**7**	These 7 bytes all have a fixed value of 55h. They wake up the receiving EMAC ports and synchronize their clocks to that of the sender's clock.
Start-of-Frame Delimiter	**1**	This 1-byte field has a fixed value of 5Dh, and immediately follows the preamble pattern indicating the start of important data.
Destination address	**6**	This field contains the Ethernet MAC address of the intended EMAC port for the frame. It may be an individual or multicast (including broadcast) address. If the destination EMAC port receives an Ethernet frame with a destination address that does not match any of its MAC physical addresses, and no promiscuous, multicast, or broadcast channel is enabled, it discards the frame.
Source address	**6**	This field contains the MAC address of the Ethernet port that transmits the frame to the Local Area Network.
Length/Type	**2**	The length field indicates the number of EMAC client data bytes contained in the subsequent data field of the frame. This field can also be used to identify the data type carried by the frame.
Data	46-1500	This field carries the datagram containing the upper-layer protocol frame (the IP-layer datagram). The maximum transfer unit (MTU) of the Ethernet is 1500 bytes. Therefore, if the upper-layer protocol datagram exceeds 1500 bytes, the host must fragment the datagram and send it in multiple Ethernet packets. The minimum size of the data field is 46 bytes. Thus, if the upper-layer datagram is less then 46 bytes, the data field must be extended to 46 bytes by appending extra bits after the data field, but prior to calculating and appending the FCS.
Frame Check Sequence	**4**	A cyclic redundancy check (CRC) is used by the transmit and receive algorithms to generate a CRC value for the FCS field. The frame check sequence covers the 60 to 1514 bytes of the packet data. Note that the 4-byte FCS field may not be included as part of the packet data, depending on the EMAC configuration.

Table 1: Description of the Ethernet frame components

1.2 BLUESPEC

1.2.1 About BSV

BSV (Bluespec SystemVerilog) is a language used in the design of electronic systems (ASICs, FPGAs and systems). BSV is used across the spectrum of applications—processors, memory subsystems, interconnects, DMAs and data movers, multimedia and communication I/O devices, multimedia and communication codecs and processors, signal processing accelerators, high-performance computing accelerators, etc. BSV is also used across markets—from low-power, portable consumer items to enterprise-class server-room systems.

Being both a very high-level language as well as fully synthesizable to hardware, it is used in many design activities, as described below. This combination of high level and full synthesizability enables many of these activities that were previously only done in software simulation now to be moved easily to FPGA-based execution (whether the design is eventually bound for ASICs or FPGAs). This can speed up execution by three to six orders of magnitude (1000x to 1,000,000x). Such a dramatic speedup not only accelerates existing activities, but enables new activities that were simply not feasible before (such as cycle-accurate multi-core processor architecture models running real operating systems and real applications).

Executable Specifications (synthesizable): For today's complex systems, a specification written in a human language (like English) is likely to be imprecise, incomplete or even infeasible and self-contradictory in its requirements. A specification in BSV addresses these concerns, because of its precise semantics and executability on real data. Fleshing out a spec in this way is also called spec validation.

Virtual Platforms (synthesizable): Today's chips are dominated by the volume and complexity of the software that runs on them. It is no longer acceptable to have to wait for chips to be available before developing software. Virtual Platforms enable software development and testing to begin as early as day one. Virtual platforms written in BSV, being synthesizable, run on FPGAs at much higher speeds and greater accuracies than traditional software virtual platforms, greatly increasing software development productivity.

Architectural Modeling (synthesizable): Complex SoCs (Systems on a Chip) involve complex trade-offs. What should be done in SW vs. in HW accelerators? How many processors vs. HW accelerators? What micro-architecture for processors, interconnect, memory systems? These decisions will affect chip area, speed, power consumption, cost, and time-to-market. Many of these decisions are too complex to be taken analytically—they require actual execution on actual data. BSV is used for synthesizable architecture models, exploration and validation at the system, subsystem, and the IP block levels.

Design and Implementation: This is the classic design activity of IP blocks (Intellectual Property), except that today these can be complex subsystems. BSV enables designing such components at a much higher level of abstraction and with better maintainability.

Verification Environments (synthesizable): Every verification environment is a model of "the rest of the system." As such, they can be as complex as designs themselves, face similar speed-of-execution issues, and face similar issues of reusability, maintainability and evolvability. BSV is used for writing synthesizable transactors, testbenches and both system and reference models.

1.2.2 Key ideas in BSV

BSV borrows powerful ideas that have been developed over several decades in advanced formal specification and programming languages. These can be classified into two groups—behavior and structure.

1.2.2.1 Behavior

BSV's behavioral model is popular among formal specification languages for complex concurrent systems. We call them Atomic Rules and Interfaces, but in the literature they also go by various names such as Term Rewriting Systems, Guarded Atomic Actions, and Rewrite Rules. Well-known formal specification languages that use a similar model include Guarded Commands (Dijkstra); TLA+ (Lamport); UNITY (Chandy and Mishra); and Z, B and EventB (Abrial).
The reasons BSV and all these formal specification languages choose this model are:

• **Parallelism:** The model of atomic rules is fundamentally parallel, unlike many other approaches that graft a parallel extension on top of a sequential language (such as those based on C++). This is ideally suited for the massive, fine-grain, heterogeneous parallelism that is everywhere in complex hardware designs and distributed systems.

• **Correctness:** The atomicity of rules is a powerful tool in thinking about correct behavior in the presence of complex parallelism. A key concept in correct systems is the invariant, which represents a correctness condition that one expects to be maintained in a system (such as, "A cache line can be in the DIRTY state in only one of the multiple caches", or "The counter in the DMA is equal to the number of bytes transferred"). Atomicity of rules allows reasoning about invariants one rule at-at-time. Without atomicity, one has to worry about all possible interleavings of parallel activities, an exercise that quickly becomes too complex both for humans and for formal verification systems.

The hardware designer sees profound practical consequences in expressing behavior using Atomic Rules and Interfaces. Any interesting piece of hardware has control logic (multiplexers, enable signals, and so on). This control logic can become extremely complex. Further, control logic typically spans module boundaries, and manifests itself at module boundaries in ad-hoc protocols and assumptions on signalling. Without a rigorous semantics, designs tend to accumulate layer upon layer of such assumptions, often poorly documented (if at all) and poorly communicated between the designer of a module and the users of the module. This is what makes RTL design so brittle, because even the smallest change can violate one of these myriad control assumptions, resulting in race-conditions, dropped data, mis-sampled data, buffer overflows and underflows, and the like. Atomic Rules and Interfaces are the solution—they provide a rigorous semantics and a much higher level view of interactions between modules, and the complex control logic is automatically synthesized, correct by construction. Even more important: when the source code is changed, a simple recompilation will re-synthesize all this complex control logic, accommodating all the changes in a correct manner. This allows the hardware designer to focus on overall architecture structure, and leave the nitty-gritty detail of control logic to the compiler (which is normally designed and maintained by hand by the RTL designer).

1.2.2.2 Structure

Modern high-level programming languages have developed some powerful abstraction mechanisms for organizing and structuring large, complex, software systems. Unfortunately, until BSV, many of these capabilities have largely passed hardware design languages by, in part under the perhaps mistaken presumption that those abstraction mechanisms are “not relevant” for hardware design languages.

BSV builds on, and borrows ideas from, two sources for its structural abstractions:

• **SystemVerilog,** for modules and module hierarchy; separation of interfaces from modules; syntax for literals, scalars, expressions, blocks, loops; syntax for user-defined types (enums, structs, tagged unions, arrays), and so on. All these will be very familiar to people who know Verilog and SystemVerilog.

• **Haskell,** for more advanced types, parameterization and static elaboration. Haskell is a modern "pure functional programming" language that is recently seeing a resurgence both because it addresses software complexity and because it is a promising basis for attacking the "parallel programming software crisis" that has been precipitated by the recent advent of multi-core and multi-threaded processors. Haskell has many features that are well acknowledged to be more powerful than those in languages like C++, Java and C#.

CHAPTER 2

Ethernet MAC controller - designing in Bluespec

The project implements the MAC protocol using Bluespec. The project implements all the following features.

2.1 IEEE 802.3 standards

In the case of Ethernet, according to 802.3-2002 section 4.1.4, the functions required of a MAC are:

- Receive/transmit normal frames
- Append/check FCS (frame check sequence)
- Half-duplex retransmission and backoff functions
- Interframe gap enforcement
- Discard malformed frames
- Prepend(tx)/remove(rx) preamble
- SFD (start frame delimiter), and padding
- Half-duplex compatibility: append(tx)/remove(rx) MAC address

The frequencies for the transmit and receive clocks are fixed by the IEEE 802.3

specification as shown below:

• 2.5 MHz at 10 Mbps

• 25 MHz at 100 Mbps

• 125 MHz at 1000 Mbps

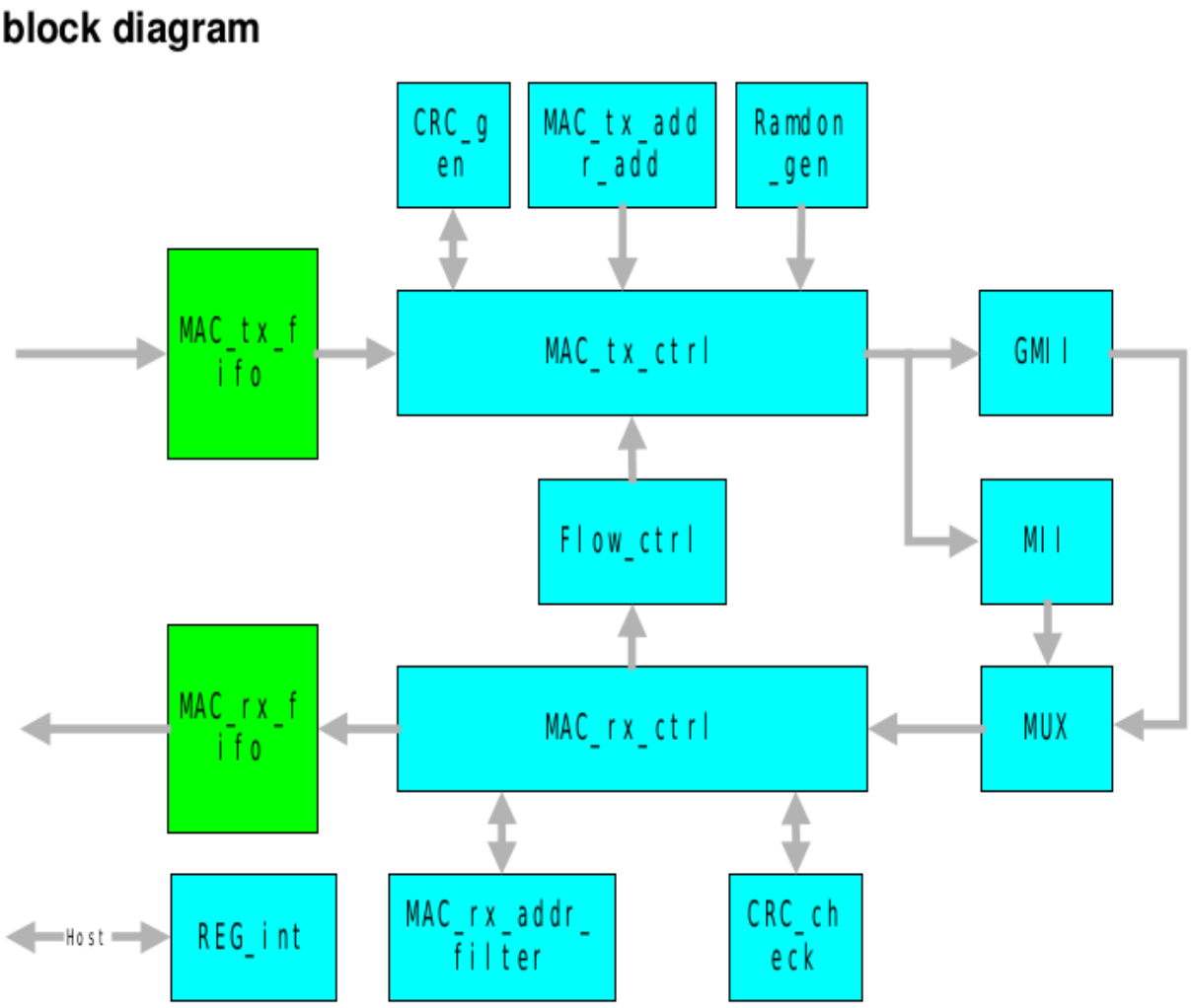


Figure 2: Block diagram of the design

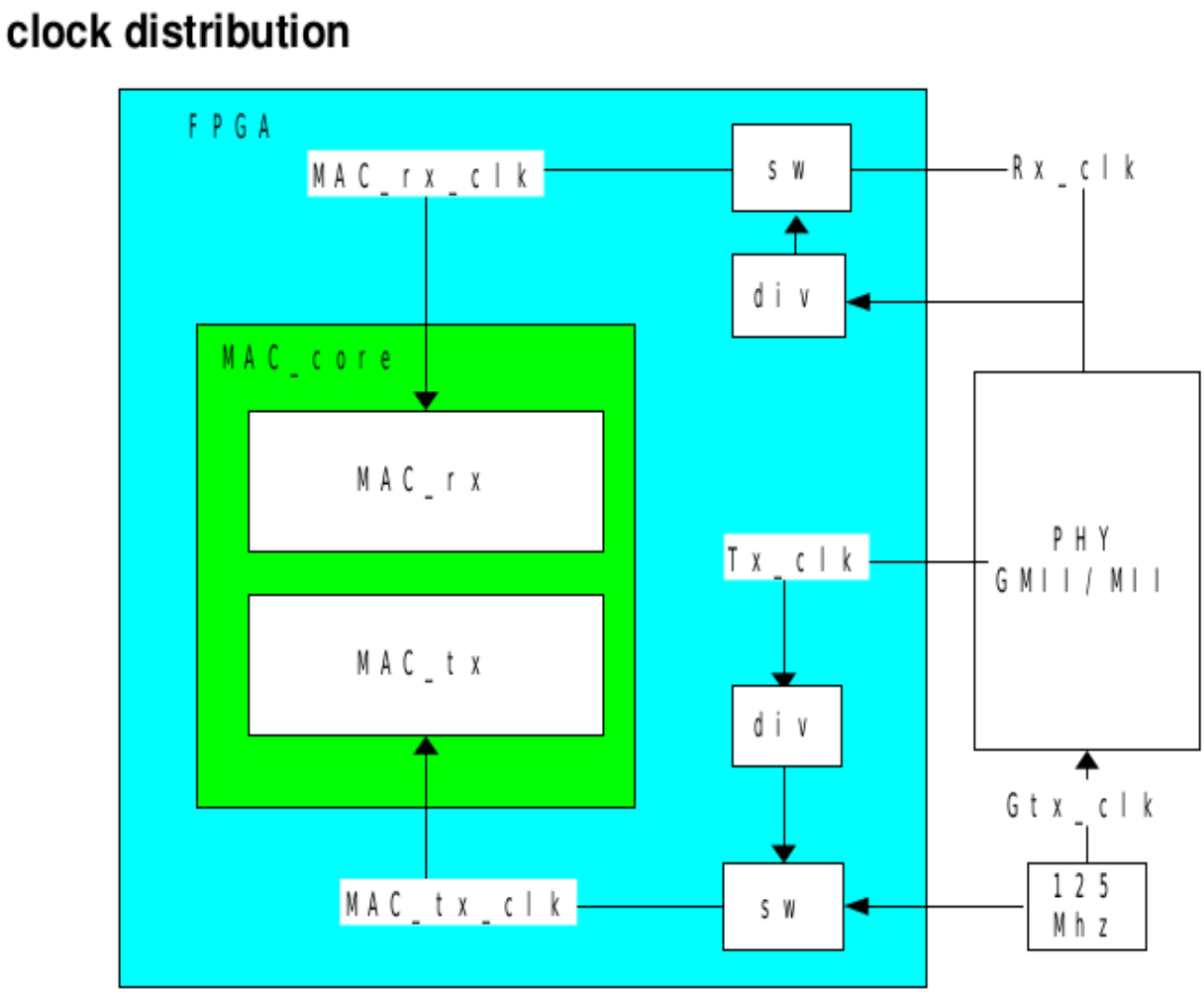


Figure 3: Block diagram of the clock distribution

2.2 Description of the design

Brief descriptions of the 4 important blocks of the design are given below. Further explanation of each of the block's design is covered in the code description.

2.2.1 MAC Receiver

The MAC receiver detects and processes incoming network frames, de-frames them, and places them into the receive FIFO. The MAC receiver also detects errors and passes statistics to the statistics RAM.

2.2.2 Receive FIFO

The receive FIFO consists of 68 cells of 64 bytes each and associated control logic. The FIFO buffers receive data in preparation for writing into packet buffers in device memory and also enable receive FIFO flow control.

2.2.3 MAC Transmitter

The MAC transmitter formats frame data from the transmit FIFO and transmits the data using the CSMA/CD access protocol. The frame CRC can be automatically appended, if required. The MAC transmitter also detects transmission errors and passes statistics to the statistics registers.

2.2.4 Transmit FIFO

The transmit FIFO consists of twenty-four cells of 64 bytes each and associated control logic. This enables the largest allowed packet (1518 bytes) to be sent without the possibility of underrun. The FIFO buffers data in preparation for transmission.

2.2.5 Water mark

The watermark described here shows how the FIFOs function while transmitting and receiving the packets:

Tx_Hwmark and Tx_Lwmark:- This two registers are used to set transmit FIFO high water mark and low water hark.When the packet data stored in transmit Fifo meet low water,transmit control logic will begin to read packet data from FIFO and send it to Phy through GMII/MII interface. In addition,hight water mark and low water mark setting is correlated with Tx_mac_wr signal. When the transmit Fifo meet high water mark , Tx_mac_wr will be asserted 0 to tell user application to hold packet transmitting.

When the transmit Fifo under low water mark, the Tx_mac_wr signal will be disasserted and the user application can proceed to transmit packet data.

Rx_Hwmark and Rx_Lwmark:- This two registers are used to set receive Fifo water mark. When FIFO received one full packet or stored packet data meet hight water mark,the Rx_mac_ra will be asserted.The Rx_mac_ra will be disasserted while FIFO below low water mark. If there is one full packet received from PHY, The Rx_mac_ra will disasserted when the whole packet is read out from Fifo.

2.3 How the code was developed?

2.3.1 2 clocks:

The MAC controller has a transmitter part and a receiver part. Both require separate clocks. Because the transmitter has its own functioning and the receiver has its own functioning. Which are exclusive with respective to each other. Though both interact with each other, both have their own individual operations to perform that are independent of each other.

The receiver and the transmitter are normally on two different circuits or two different chips. So they should normally be implemented using two different clocks. In this code I have put the transmitter and the receiver both on the same module. But if we want to manufacture the circuits and send it for fabrication then the two circuits Transmitter and the receiver can be separated and sythesized. The module that I have designed implements the functionality of the MAC protocol.

Exposecurrentclock is the interface that will create clocks for the module. Along with a clock there is always a reset. Some of the registers will be clocked by one clock and the rest by another clock. The transmitter side registers will be clocked by the transmitter clock and the receiver side registers with the receiver clock.

2.3.2 FIFOF package:

This package helps to instantiate a level FIFO. This FIFO gives the level of the packets in the FIFO. It has two functions isGreaterThan() and isLesserThan(). These functions take in an integer argument and returns a Bool value showing whether the level in the FIFO is greater or lesser than the values given in the arguments.

I have used these functions to implement the watermark in the FIFOs. This is important because the transmitter keeps getting packets that need to be transmitted across to the receiver from the user continuously. But the transmitter is limited in its capacity. Depending on the congestion in the PHY medium and many other parameters it will be able to send the packets one by one slowly. But in the mean time the FIFO gets filled up with all the packets from the user.

The water mark is a very wonderful implementation. As the name suggests it is like a watermark in the overhead tank in our houses. In case we have an automatic switch to fill the water in, the system pours water into the tank when the level goes below the lower water mark and stops when the level reaches the high water mark it switches off the motor.

It allows the user to transmit the packets to the transmitter FIFO when the level of the FIFO is less than the lower water mark. As it keeps accumulating then when it crosses the high watermark then it tells the user to wait till the time the FIFO level reduces to the lower water mark. As soon as the level goes below the lower watermark then again it allows the user to send the packets to be transmitted.

The FIFOs available in Bluespec is of a great advantage. If we have to implement this in verilog we will have to separately write a module for the FIFO and it becomes very tough. Here it is readily available. The 100 lines of code in verilog could be written in a single line instantiating the FIFO. Moreover the special FIFOs and the level FIFO available in Bluespec make it very easy to handle. Designed according to the needs of the engineers.

2.3.3 Config reg

Used to get the packets (implementation of user interface) this is taking the packets from a file into the code.

I have tried to implement the USER INTERFACE through the Config Reg package. This package helps to read data from a file – this is like a user trying to transmit packets and asking the transmitter to transmit.

The data will be loaded in a register file kind of interface where the data is stored in a kind of register array with indices. The regFile module has few methods one of which is the sub() method. This method takes in an integer value and gives out the data corresponding to the integer value in the register array.

2.3.4 Parameters

The parameters that I have used are fifo depth and width. Fifo width is 8 bits. Since normally the transmission is done in octets of 8 bits each. The Fifo depth I have used is 512. It can be changed also. This is another user friendly function available in Bluespec called Parameterization. You can have a separate file and give in all the parameters and their values.

If you want to change some parameters then you just have to change the values in this file.

This will automatically get communicated to the other files like the main module and the test bench also.

According to the IEEE 802.3 standards the normal payload length will vary from 46 packets to 1532 packets. We can change the FIFO depth accordingly. If we are sending larger packets then we can increase the FIFO depth and if we are going to send only the small packets then we can have a smaller FIFO depth.

2.3.5 Checksum – verification

Check sum is a very important function in the MAC protocol. When the transmitter sends the packets the transmitter calculates the checksum of all the data present in the packet. It generates a 32 bit number called the CRC32 or FCS which is appended to the frame that is sent.

When the receiver gets the packets it performs a circular checksum of all the data in the frame including the FCS also in the checksum calculation. The checksum function is designed in such a way that when both times it is done as above it generates a magic number which is 0xC704DD7B in hexadecimal format.

The receiver checks if the complete checksum is coming out to this value or not. If it comes to this value then the received packets are correct. There was no error while transmission. And the packet sends the next one and so on. If it finds our that it is not equal then the receiver sends an error request to the transmitter and asks to resend the packet.

I have implemented the checksum with a C file. Since this is a complicated function but still it is widely used the code for the function is readily available in C language. Since Bluespec has this good facility to insert a C file into the code. I had taken advantage of this in my core. The function takes in a 32 bit number and the 8-bit data and does a type of series of xor functions in a circular fashion which is quite complex to figure out easily.

All the packets one after the other are given into this function as separate octets (8-bit data). The result of the function is again given back in the field of the 32 bit number taking the the next octet into the 8-bit data input into the function. At the end when the data comes to the end of the frame the final value returned will be the checksum value.

This preserves the data that is sent. Every kind of data will produce a unique checksum. And this unique checksum when sent to the receiver side goes in as the octets one after the other finally gives the MAGIC NUMBER (normally called) 0xC704DD7B.

The inserting of the C program in Bluespec is a wonderful function available in Bluespec. This is not normally available in languages like Verilog or VHDL. Where one has to code at the level of the registers which is a very pain staking task. And makes the whole design very complex and difficult for the designer to understand.

But Bluespec is at an abstract level but it is more user friendly while designing.

2.3.6 dest_mac_add() and source_mac_add()

This is a method that I have used to read the destination MAC address for the receiver. While transmitting the data the transmitter appends the destination MAC address to the packets that are being sent. When the receiver receives the packets it checks whether the destination MAC address matches with that of the receiver then it starts reading the further packets.

Similarly the source MAC address is also sent along with the actual frame.

2.3.7 Frame generation:

As mentioned above in the Ethernet MAC protocol section the main data packets that have to b transmitted should be appended with the MAC destination address, MAC source address, the payload(the main data), FCS. This is the frame structure.

I have used 3 register sentPacket, sentFCS and sentPayload to design the transmission of the frames.

The logic that I used is normally the sentPayload option will be false or 0 indicating that the payload has not been transmitted. After the destination and the source MAC address is sent then the Payload is sent. Once the pay load is sent the sentPayload register turns to True or 1. Indicating to the transmitter that the FCS has to be sent now. Now the sentFCS register is still False. Once the FCS is sent then the sentFCS register becomes True. When this happens the packet frame is completely sent. Now the sentPacket becomes True.

This will indicate to the transmitter to send the IPG (Inter packet Gap).

2.3.8 Interpacket Gap (IPG or IFG):

The interpacket gap is normally 96bits. Or in terms of time it is 96 times the time taken to send one bit. In this case it is equal to 12 octets. So for 12 octets the transmitter doesnt send any data. This is the time that is given to the receiver to verify the data that has come and also to tune on the clock of the receiver, to synchronize it and be ready to receive the next packet.

2.4 “Rules” in Bluespec

I would like to mention a little bit about the rules in Bluespec here:

BSV does not have always blocks like Verilog. Instead, rules are used to describe all behavior (how state evolves over time). Rules are made up of two components:

• Rule Condition: a boolean expression which determines if the rule body is allowed to execute (“fire”)
• Rule Body: a set of actions which describe the state updates that occur when the rule fires

As in many languages, there is a logical view or semantic view of BSV behavior which is distinct from the implementation. The former is how designers/programmers think about their source code. The latter is the output of the compiler (in our case, Verilog or Bluesim) and, ideally, should not be of concern to someone trying to explain or understand a BSV program.

In BSV, the logical sequence of rule executions are further grouped into sub-sequences called “clocks”. There is no a priori limit on how many rules can fire within a clock. However, instead of allowing arbitrary sub-sequences of rules within a clock, we impose some constraints based on pragmatic hardware considerations. These constraints are:

- Each rule fires at most once within a clock.
- Certain pairs of rules, which we will call conflicting, cannot both fire in the same clock.

Conflicts arise due to hardware considerations. For example, in hardware each state element can change state only once per clock. Thus, we cannot have a rule sub-sequence within a clock where one

rule's state update is read by a later rule in the same clock. We call this a "rule ordering conflict". Similarly, certain hardware resource constraints, such as the fact that a hardware wire can only be driven with one value in each clock, will preclude some pairs of rules from executing together in a clock because they both need that resource. We call this a "rule resource conflict".

2.5 Rules that are implemented in the code:

2.5.1 Enable transmission rule

The first rule that I have implemented is the enable transmission rule.

The transmitter transmitts the packets to the receiver when the receiver FIFO packet level is less than the lower water mark and it stops the transmission when the packet level is above the high water mark. This is done through the register named enable_transmission which becomes 1 to indicate to trasmit and vice versa.

2.5.2 Send preamble and SFD rule

The second rule that I have implemented is the sending preamble and the SFD.

The preamble is a series of 0's and 1's that are sent to the receiver to indicate that in some time the packet is going to come and indicating the receiver to get ready to receive. In this time the receiver gets the clock tuned and gets ready to receive the packets. If first receives the destination MAC address and then the source MAC address and then the payload.

After the preamble there is something called as the SFD which means Start Frame Delimiter. This is to indicate to the receiver that the preamble is being stopped and the packet is going to start. The value of this is 0xAB in hexadecimal system

2.5.3 Transmit octets

The next rule is to transmit the main data in the form of octets. This will be done as soon as the SFD is being sent. While the octets are being sent simultaneously the FCS is being calculated with every octet going in as the 8-bit input into the function of the checksum. The returned value of the checksum goes in as the 32 bit input for the next octet checksum calculation. For the initial octet the 32 bit input will be 0xFFFFFFFF.

Once the transmitter FIFO gets empty which means that the payload is transmitted then the FCS is transmitted immediately. The FCS is 32 bit wide data so it is transmitted in 4 octets.

As soon as the FCS is sent the IPG is transmitted. To give a gap for the next frame to be sent.

This completes the transmitter side functioning. Which is the transmitter FIFO and the control logic as described in the rules.

Normally in verillog to implement this will take a very very long code which is very cumbersome to handle. But bluespec makes it very easy. With the rules etc.

2.5.4 Analyzing the received packets

Now we come over to the Receiver's side:-

I will now explain the implementation of the receiver FIFO and the corresponding control logic of it.

For the receiver I have used few rules and some registers that will implement its functioning.

The packets in the form of the octets accumulate in the receiver FIFO. This receiver logic will analyze the packets and identify the payload and also verify whether the packets have been transmitted without any error which is done through the CRC checksum function again.

I have used 2 registers reg1 and reg2 which will detect the SFD. Each one is of 8 bits size. The condition would be if (reg1 == 8'hab && reg2 == 8'haa) which will tell that the SFD has been transmitted and then the destination MAC address is going to come is going to come.

The MAC address is 6 octets long. This is stored in a register array. Creg is the module that can create register arrays of a particular length. Once the dest MAC address is ready then through the mac_dest_rdy register it indicates to the rule in the test bench saying that the MAC destination address is avaiable. The same thing is done for the source MAC address also in a similar manner.

Once the souce MAC address is also read from the receiver, the payload is now read. The first 2 octets of the payload indicate the length of the payload. This is called the EtherType. Sometimes it is also used to indicate the type of packet that is being transmitted like the ADP or the UDP etc.

Once the receiver FIFO becomes empty the control logic of the receiver comes to an end.

While the payload is being read in as octets till the receiver FIFO becomes empty simultaneously the FCS is also being calculated. But this time it is also including the 4 octet checksum appended by the transmitter. Towards then end of the receiver control logic the FCS is also calculated. It is checked whether the value of the new FCS is equal to the MAGIC NUMBER 0xC704DD7B.

2.6 References:

www.opencores.com

I have used the IP core of Trimode-ethernet MAC (TEMAC) designed in verilog by Jon Gao. This was available on opencores. This core helped me to design the MAC.

CHAPTER 3

Conclusions

1. From the project I understood that Bluespec is a very user friendly language to design circuits.
2. It has a variety of concepts that helps in designing very robust and compact circuits.
3. The default packages available in Bluespec reduce the effort from the user side in designing basic functional units.
4. Also it helps to implement large scale circuits in just a few lines.
5. A good feature in Bluespec is that most of the modules are syhthesizeable.
6. You can design at a very abstract level the functioning of a circuit and it will still synthesize it.
7. I feel that Bluespec can create a break through in the current circuit design technology in designing circuits that were thought to be difficult to synthesize in the existing languages like verilog etc.
8. It can reduce the designing time and the debugging time.
9. It can bring out new features immediately in the existing technologies because of its abstract level and user friendly features.