=========================================================================

ATTENTION !!

Справочная информация по функциям Ethernet-устройств и форматам пакетов, необходимых для поддержания UDP-протокола :

\* О запросе-ответе для установления связи по IP-адресам с помощью ARP :

1. https://ru.wikipedia.org/wiki/ARP (рус)
2. https://en.wikipedia.org/wiki/Address\_Resolution\_Protocol (en, более подробная диаграмма ARP-пакета)
3. https://tools.ietf.org/html/rfc826 (en, описана структура ВСЕГО пакета, но без конкретных значений полей).

------------------------------------------------------------------------------

Ни один урод не описал НОРМАЛЬНО всю структуру пакета, содержащего ARP-запрос и ARP-ответ. Из нескольких источников удалось по крохам собрать следующую информацию:

Общая структура всего Ethernet-пакета в целом описана в [3].

Ethernet transmission layer (not necessarily accessible to the user):

(преамбула c StartFrameDelimiter ) <- прим.Yu

Ethernet header :

48bit: Ethernet address of destination <- MAC-addr (прим.Yu)

( REQUEST как раз и запрашивает этот MAC-addr, поэтому он рассыла-

ется broadcast-пакетом, Daddr = H"FF FF FF FF FF FF" ;

А вот REPLY посылается конкретно тому, кто присылал REQUEST,

поэтому Daddr = ( Saddr, полученный в REQUEST-е) ).

48bit: Ethernet address of sender <- MAC-addr (прим.Yu)

16bit: Protocol type = H"0806" <- упомянуто в [1]

Ethernet packet data:

16bit: Hardware Type ( Ethernet = H"0001" ) <- см. [2]

16bit: Protocol Type ( IPv4 = H"0800" ) <- см. [2]

8bit: Length of Hardware\_Addr (т.е. MAC-addr => H"06") <- см. [2]

8bit: Length of Protocol\_Addr (для IP = 4Bytes => H"04") <- см. [2]

16bit: OpCode ( REQUEST = H"0001" | REPLY = H"0002" ) <- см. [2]

(6)Bytes: Hardware\_Addr of sender of this\_packet <- MAC-addr (прим.Yu)

(4)Bytes: Protocol\_Addr of sender of this\_packet <- IP-addr (прим.Yu)

(6)Bytes: Hardware\_Addr of target of this\_packet

(в REQUEST-е этот адрес как раз запрашивается, поэтому

это поле = H"00 00 00 00 00 00" )

(4)Bytes: Protocol\_Addr of target (т.е., of Destination) <- (прим.Yu)

Ethernet FrameCheckSequence (т.е., CRC) <- (прим.Yu)

=========================================================================

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Приём Ethernet-фреймов (посылок) всех типов производится модулем ETHERNET. В данном разделе кратко описано взаимодействия этого модуля с MainProject-ом.

=========================================================================

TITLE "ETHERNET";

Модуль ETHERNET выполняет приём и передачу "Raw\_Ethernet"-фреймов. Во время приёма фрейма модуль проверяет значение поля "PacketLength/Type" и, в зависимости от него, действует следующим образом:

* если значение < 1536 => это фрейм "Raw\_Ethernet", и значение равно размеру Блока Данных (в Байтах) принимаемого фрейма. Тогда указанное число байт принимается, каждая пара Байт передаётся в виде 16-битного слова в MainProject.
* если значение > 1536 => это фрейм НЕ "Raw\_Ethernet", поэтому значение поля "PacketLength/Type" является кодом типа пакета, содержащегося в БлокеДанных принимаемого фрейма. Тогда значение поля "PacketLength/Type" передаётся по выходам RxLenTyp[16..1] в MainProject для дальнейшего распознавания и определения размера БлокаДанных, который нужно принять в этом фрейме. Модуль ETHERNET начинает принимать данные БлокаДанных, при этом размер БлокаДанных, определёный MainProject-ом, должен быть передан в модуль ETHERNET (по входам RxLengIn[12..1]) не позже 20-го байта. После этого модуль ETHERNET принимает остаток данных до указанного числа байт. =Все= принятые данные передаются в MainProject, каждая пара Байт передаётся в виде 16-битного слова.

Для передачи пакета данных MainProject должен записать данные, которые нужно будет передать в поле данных пакета, в соответствующую область специального ОЗУ (предназначенного для работы с Ethernet-пакетами), а также должен записать значения всех необходимых параметров пакета в соответствующие регистры.

Например, для передачи UDP-пакета необходимо:

* записать данные, которые должны быть переданы в БлокеДанных UDP-пакета, в ОЗУ, начиная с addr=8254 (62-я ячейка ОЗУ).
* обеспечить, чтобы данные UDP-хедера могли быть прочитаны модулем ETHERNET в нужные моменты. Для этого эти данные должны быть записаны в определённые адреса ОЗУ либо в соответствующие регистры, и данные из этих источников нужно подавать в правильном порядке (конкретный порядок следования данных для разных типов исходящих пакетов см. ниже в описании Передачи соответствующего типа пакета) на входы DataBus\_in[15..0] модуля ETHERNET.
* подать значение размера БлокаДанных, который нужно отправить, на входы TxLength[12..1] модуля ETHENET.
* подать код типа пакета ("IP" <=> H"0800") на входы TxLenTyp[16..1] модуля ETHERNET.

ПАРАЛЛЕЛЬНО с выполнением приёма Ethernet-фрейма все его байты, начиная с поля DestinationAddr, =попарно= выдаются в MainProject в виде 16-битных слов данных, синхронизованных с =внутренним Clock-ом MainProject-а=. Каждое слово "распутывается" и защёлкивается в регистр Reg\_RxWord. Сразу после защёлкивания очередного DWord в Reg\_RxWord выдаётся запрос на вычитывание готового слова в MainProject в виде (IORq=1, RW=0). По этому запросу слово данных может быть записано в отдельное ОЗУ секции "ETHERNET Interface" основного проекта.

------------------------------------------------------------------------------

Подробное описание модуля ETHERNET приведено в файле "ETHERNET\_4.txt". НИЖЕЛЕЖАЩИЙ ТЕКСТ содержит подробное описание приёма и передачи всех типов пакетов, которые необходимы для обмена данными между (пользовательским) Устройством и Компьютером, на котором работает =стандартный= софт.

Кроме того, в главе "ТЕСТИРВАНИЕ приёма/передачи данных" описаны узлы, с помощью которых реализуются специальные режимы работы всего Устройства, предназначенные для тестирования приёма/передачи данных по протоколу UDP. Эти режимы обеспечивают возможность проведения тестов с помощью Компьютера, на котором работает =стандартный= софт под Windows.

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

=========================================================================

ПРИЁМ ПАКЕТОВ различных типов.

----------------------------------

После того, как принятое значение "PacketLength/Type" защёлкнулось в регистр RxLenTyp модуля ETHERNET, оно остаётся неизменным до окончания приёма всего фрейма. Значение RxLenTyp[] поступает в MainProject на логику распознавания Типа пакета. Если это значение соответствует одному из опознаваемых типов ("CMD3", "IP", "ARP"), то флаг опознанного типа устанавливается в =1, и инициируется процедура обработки соответствующего протокола.

"CMD3" :

--------

Логика распознавания типа пакета выставляет TypeCMD3=1 и значение размера пакета Ether.RxLengIn[]=64(Байта).

Когда флаг Ether.RxDataBlck переключается в =1, разрешается работа счётчика EtherAdrCt (на его входе .sclr становится =0). (ЕСЛИ БУДЕТ НУЖНО: По сигналу EtherInit этот счётчик загружает в себя начальный адрес (=??) для записи данных). По запросу (Ether.IORq=1, Ether.RdWr\_out=0) очередное слово данных DWordNN записывается в EtherRAM (через PortA). Через 1такт Clk100 счётчик EtherAdrCt инкрементируется, и содержит адрес для записи следующего слова данных.

При выполнении процедуры протокола CMD3 стробы ProtoNext и ProtoNext3 не вырабатываются => счётчик EtherAdrCt просто инкрементируется до окончания приёма блока данных.

"ARP" :

-------

Логика распознавания типа пакета выставляет TypeARP=1 и значение размера пакета Ether.RxLengIn[]=46(Байт). Появившийся сигнал TypeARP=1 инициирует "машину состояний" протокола ARP (по следующему \_/ RxClk триггер RxARPheader переключается в =1), а также вводит счётчик Протокола CtProto в режим счёта. Хронограмма основных сигналов, возникающих во время обработки протокола ARP, приведена ниже на рис.1.

Все 6+6+2=14Байт=7DWords Ethernet-хедера записываются в ОЗУ EtherRAM в addr=8192+[0..6]. По готовности каждого слова модуль ETHERNET выдаёт запрос на вычитывание из себя данных (IORq=1, RW=0). По этому запросу формируется строб EthWord\_Rd, по которому слово данных записывается в текущий адрес ОЗУ EtherRAM. На следующем такте Clk100 из EthWord\_Rd вырабатывается строб EthAdrCt\_en, по которому счётчик адреса инкрементируется (или в него загружается начальный адрес для записи следующей порции данных, относящихся к следующему полю ARP-пакета).

Для "облегчения счёта" все слова DataBlock-a решено записывать в память начиная с addr=8192+8=8200. Поэтому =после= записи слова "PacketLengh/Type" (в addr=6) И =до= записи 1-го слова DataBlock-a нужно загрузить в счётчик адреса EtherAdrCt значение =8. Для этого во время, когда принимаются первые2Байта DataBlock-а, по условию Ether.RxDataCt[]==1 (т.е. заведомо =до= записи 1-го слова DataBlock-a, и даже до записи слова "PacketLength/Type"), простенькая логика выдаёт сигнал RxAdrSet2 (по \_/^ RxClock); с помощью триггеров RxAdrSet2a и RxAdrSet2b срабатывание этой логики гарантированно пере-привязывается к Clk100, и при RxAdrSet2b=1 (в течение 4\*Clk100) вырабатывается сигнал EthAdrInit и устанавливает значение =8 на входах "регистра" EthNextAdr. По сигналу EthAdrInit схемка на триггерах EthAddrLd1 и EthAddrLd2 вырабатывает строб, по которому начальный адрес защёлкивается в "регистр" EthNextAdr, причём RS-триггер EthAddrLd2 остаётся в =1. Поэтому, когда после записи слова "PacketLengh/Type" вырабатывается строб EthAdrCt\_en, счётчик EtherAdrCt загружает в себя адрес из "регистра" EthNextAdr (а не инкрементируется), а триггер EthAddrLd2 сбрасывается в =0 (поэтому при последующих стробах EthAdrCt\_en, пока снова не будет инициализирована =загрузка= адреса, счётчик EtherAdrCt будет инкрементироваться). Таким образом, схема приведена в готовность к приёму DataBlock-a, содержащего ARP-запрос.

Последующие поля ARP-запроса нужно записать в ОЗУ в таком порядке, чтобы при передаче ARP-ответа можно было просто вычитывать даные из ОЗУ последовательно друг за другом. Ещё раз подчеркнём, что порядок расположения данных в этом куске ОЗУ "заточен" под ARP-ОТВЕТ!!

Данные располагаются в ОЗУ в следующем порядке :

- "HardwareType": Ethernet = H"0001" addr= 8 ;

- "Protocol" : IP = H"0800" addr= 9 ;

- "Length of HWare\_addr | Length of Protcl\_addr" = H"0604" addr= 10 ;

- "Operation Code: ARP-Rq = H"0001" | ARP-reply = H"0002" addr= 11 ;

Число H"0002", которое нужно вычитывать каждый раз при от-

правлении ARP-ответа, вписано в этот адрес при инициализа-

ции Альтеры и должно НЕ ИЗМЕНЯТЬСЯ. Чтобы не записать в

этот адрес другое число, слово H"0001", полученное при при-

ёме ARP-запроса, записывается в "ненужную" ячейку addr=31.

- "SourceAddr"+"IP-Saddr" (6+4=10Байт => 5 16-битных слов) addr=[12..16];

В ARP-ответе это MAC-адрес и IP-адрес Этого\_Устройства со-

ответственно. Эти данные вписаны в эти адреса при инициа-

лизации Альтеры и должны НЕ ИЗМЕНЯТЬСЯ. Чтобы не записать

в эти адреса другие числа, данные поля "DestinationAddr"+

+"IP-Daddr", полученные при приёме ARP-запроса, записыва-

ются в "ненужную" область памяти.

- "DestinationAddr"+"IP-Daddr" (6+4=10Байт => 5 16-бит.сл.) addr=[17..21];

В ARP-ответе это MAC-адрес и IP-адрес Компутера соответст-

венно. Эти данные записываются в эти адреса при приёме

полей "SourceAddr"+"IP-Saddr" ARP-запроса.

- "хвост" пакета (нули), дополняющий длину до 46Байт (min) addr=[22..27].

Первые 6Байт (3 16-битных слова) ARP-хедера записываются в EtherRAM друг за другом, по мере того как модуль ETHERNET принимает эти данные и выдаёт их в MainProject. По запросу (Ether.IORq=1, Ether.RdWr\_out=0) очередное слово данных DWordNN записывается в EtherRAM (через PortA, по стробу EthWord\_rd). Через 1такт Clk100 после строба EthWord\_rd счётчик EtherAdrCt инкрементируется, и содержит адрес для записи следующего слова данных.

Рис.1. Хронограмма сигналов в характерные моменты обработки ARP-запроса.

--=========================================================================

--\*\*\*\*\*\*\*\*\*\*\*\* EXECUTION of ARP-Protocol : receiving of ARP-packet \*\*\*\*\*\*\*\*\*\*

25MHz RxClk \_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_\_\_

CtReceiv.q[] ==X====12=====X====13=====X====14=====X====15=====X====16=====X====17=====X====26=====X====27====

CtDatBlk.q[] ==X====13=====X====14=====X====15=====X====16=====X====17=====X====18=====X====27=====X====28====

Reg\_RxWord[] ===========DWord03=========================XXX==========DWord04=UDP-OpCode=================XXX=DWord07

EthWord\_Rd \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_///^^\\\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_///^^\\\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CtProto.q[] ==X====12=====X====13=====X====14=====X====15=====\\_\_\_\_\_0\_\_\_\_\_/=====1=====X=====2=====X=====3====

ProtoNext \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/^^^^^^^^^^^\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

EthAdrInit \_\_\_/^^^^^^^^^^^\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/^^^^^^^^^^^\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

EthAddrLd2 \_\_\_\_\_\_\_\_\_\_///^^^^^^^^^^^^^^^^^^\\\\_\_\_\_\_\_\_\_\_\_\_\_///^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\\\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

EtherAdrCt.q[]->PortA========10==============XXX======================31====================XXX====12==========

RxARPheadr ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

RxARPSaddr \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Последующие поля ARP-запроса нужно записать в ОЗУ согласно вышеперечислен-

ному порядку адресов. Поэтому после записи 3-рёх слов ARP-хедера в адреса

addr=[8..10], перед записью 4-го слова нужно установить адрес =31. Для этого

перед записью 3-го слова ARP-хедера в ОЗУ (по условию CtProto.q[]==12) логика

состояния "RxARPheadr" выдаёт сигнал RxAdrSet3 (по \_/^ RxClock); с помощью

триггеров RxAdrSet3a и RxAdrSet3b срабатывание этой логики гарантированно

пере-привязывается к Clk100, и при RxAdrSet3b=1 (в течение 4\*Clk100) вырабаты-

вается сигнал EthAdrInit и устанавливается значение =31 на входах "регистра"

EthNextAdr. По сигналу EthAdrInit схемка на триггерах EthAddrLd1 и EthAddrLd2

и "регистр" EthNextAdr подготавливаются к тому, чтобы загрузить число =31 в

счётчик EtherAdrCt. Поэтому после записи 3-го слова ARP-хедера в ОЗУ счётчик

EtherAdrCt загружает в себя addr=31 из "регистра" EthNextAdr (а не инкременти-

руется), а триггер EthAddrLd2 сбрасывается в =0 (поэтому при последующих стро-

бах EthAdrCt\_en, пока снова не будет инициализирована =загрузка= адреса, счёт-

чик EtherAdrCt будет инкрементироваться). Таким образом, 4-е слово ARP-хедера

будет записано в addr=31.

После ARP-хедера поступают поля "SourceAddr"+"IP-Saddr" (6+4=10Байт =

= 5DWords). Данные этих полей нужно записать в ОЗУ последовательно друг за

другом в адреса addr=[17..21]. Для этого перед записью последнего слова

ARP-хедера в ОЗУ (по условию CtProto.q[]==15) логика состояния "RxARPheadr"

выдаёт сигнал RxAdrSet4, по которому в счётчик EtherAdrCt загружается число

=17 аналогично тому, как это делалось перед записью 3-го слова ARP-хедера.

Поэтому 1-е слово поля "SourceAddr" записывается в addr=17, а после записи

каждого слова данных по стробу EthAdrCt\_en счётчик EtherAdrCt инкрементируется

(пока снова не будет инициализирована =загрузка= адреса). Таким образом,

данные полей "SourceAddr"+"IP-Saddr" записываются в последовательно друг за

другом в указанные адреса ОЗУ.

Затем поступают поля "DestinationAddr"+"IP-Daddr" (6+4=10Байт=5DWords).

Эти данные ARP-запроса не нужны, поэтому они записываются в "ненужную" область

ОЗУ. Потом в эту же "ненужную" область записывается "хвост" ARP-запроса

(нули), при этом данные "DestinationAddr"+"IP-Daddr" могут быть "затёрты".

"IP" :

-------

Логика распознавания типа пакета выставляет TypeIP=1 и подключает входы

Ether.RxLengIn[] к выходам "регистра" IPlength, в который будет защёлкнуто

значение поля "IP\_Length". Появившийся сигнал TypeIP=1 инициирует "машину

состояний" протокола IP (по следующему \_/ RxClk триггер RxIPheader переключа-

ется в =1), а также вводит счётчик Протокола CtProto в режим счёта. Хроно-

грамма основных сигналов, возникающих во время обработки протокола IP, приве-

дена ниже на рис.2.

Все 6+6+2=14Байт=7DWords Ethernet-хедера записываются в ОЗУ EtherRAM

в addr=8192+[0..6]. Это делается так же, как при приёме ARP-запроса.

А вот данные IP-шного DataBlock-a нужно записывать в другой кусок ОЗУ, чем

данные ARP-пакета. Размер ARP-пакета = 46Байт=23DWords => он занимает адреса

[8..30] => адреса начиная с addr=32(=8224) пока свободны ; однако, решено

зарезервировать addr=[32..47] для хранения неизменных данных, относящихся к

IP-хедеру и UDP-хедеру, а IP-шный DataBlock принимаемого пакета записывать

начиная с addr=48(=8240). Поэтому =после= записи слова "PacketLengh/Type"

(в addr=6) И =до= записи 1-го слова DataBlock-a нужно загрузить в счётчик

адреса EtherAdrCt значение начального адреса для записи IP-шного DataBlock-a.

Это делается так же, как при приёме ARP-запроса.

Первые 12Байт (6 16-битных слов) IP-хедера записываются в EtherRAM друг за

другом, по мере того как модуль ETHERNET принимает эти данные и выдаёт их

в MainProject. По запросу (Ether.IORq=1, Ether.RdWr\_out=0) очередное слово

данных DWordNN записывается в EtherRAM (через PortA, по стробу EthWord\_rd).

Через 1такт Clk100 после записи слова счётчик EtherAdrCt инкрементируется, и

содержит адрес для записи следующего слова данных.

\*\* В течение того такта RxClk, в котором CtProto.q[]=7, модуль ETHERNET

выдаёт 2-е слово данных, являющееся значением "IP\_Length" (см. рис.2). Это

значение нужно запомнить и затем передать в модуль ETHERNET, чтобы он принял

заданное число байт. Для этого по условию CtProto.q[]=8 выдаётся строб на

RxIPleng[].ena, и по следующему \_/ RxClk значение "IP\_Length" защёлкивается в

"регистр" RxIPleng, выходы которого, как упомянуто выше, подключены к входам

Ether.RxLengIn[]. Модуль ETHERNET защёлкивает значение "IP\_Length" в свой

"регистр" RxLength и использует как признак окончания блока данных.

\*\* В течение того такта RxClk, в котором CtProto.q[]=19, модуль ETHERNET

выдаёт 5-е слово данных, старший байт которого является значением Протокола

пакета, заключённого в IP-пакете. ЕСЛИ MSByte=H"11" => это UDP-пакет, тогда

триггер TypeUDP устанавливается в =1 ; если нет, то ничего не происходит.

Приём IP-хедера продолжается.

Когда заканчивается приём первых 12Байт IP-хедера, CtProto.q[] принимает

значение =23, логика "Машины состояний" выдаёт сигналы, согласно которым по

следующему \_/ RxClk происходит :

- триггер RxIPheader переключается в =0 ;

- триггер RxIP\_Saddr переключается в =1 ;

- счётчик CtProto обнуляется (для отсчёта следующей стадии Протокола).

Рис.2. Хронограмма сигналов в характерные моменты обработки IP-пакета.

--==============================================================================

--\*\*\*\*\*\*\*\*\*\*\*\* EXECUTION of IP-Protocol : receiving of IP-packet \*\*\*\*\*\*\*\*\*\*\*\*

25MHz RxClk ^^^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^^ \_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_\_\_

CtReceiv.q[] ===7=====X=====8=====X=====9=====X===== ==X====23=====X====24=====X====25=====X====26=====X====27====

CtDatBlk.q[] ===8=====X=====9=====X====10=====X===== ==X====24=====X====25=====X====26=====X====27=====X====28====

Reg\_RxWord[] ==XXX======DWord02=IPlength============ =======XXX==========DWord06============================XXX=DWord07

EthWord\_Rd \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_///^^\\\\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_///^^\\\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CtProto.q[] ===7=====X=====8=====X=====9=====X===== ==X====23=====\\_\_\_\_\_0\_\_\_\_\_/=====1=====X=====2=====X=====3====

ProtoNext \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_/^^^^^^^^^^^\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

EthAdrInit \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_/^^^^^^^^^^^\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

EthAddrLd2 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_///^^^^^^^^^^^^^^^^^^^^^^^^^^\\\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

EtherAdrCt.q[]->PortA=============09============XXX== =============13========================XXX====16=============

RxIPleng\_ld \_\_\_\_\_\_\_\_\_\_/^^^^^^^^^^^\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

RxIPleng[].q \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/====IPlength===== =============================================================

RxIPheader ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ ^^^^^^^^^^^^^^^\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

RxIP\_Saddr \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Следующим поступает 4-Байтовое значение "IP\_Saddr", которое выдаётся в

MainProject в виде двух слов. Они записываются в addr=[54..55], (если данные

IP-пакета размещаются начиная с addr=48).

Когда заканчивается приём 4Байт "IP\_Saddr", CtProto.q[] принимает значение

=7, и логика "Машины состояний" выдаёт сигналы, согласно которым по следующему

\_/ RxClk происходит :

- триггер RxIP\_Saddr переключается в =0 ;

- триггер RxIP\_Daddr переключается в =1 ;

- счётчик CtProto обнуляется (для отсчёта следующей стадии Протокола).

Следующим поступает 4-Байтовое значение "IP\_Daddr", которое обрабатывается

аналогично и записывается в addr=[56..57], (если данные IP-пакета размещаются

начиная с addr=48).

Параллельно с записью в ОЗУ 1-е и 2-е слова "IP\_Daddr" подаются на схему

сравнения, которая сравнивает их с 1-й парой и 2-й парой байт IP-АдресаЭто-

гоУстройства, соответственно. Результат сравнения 1-го слова с 1-й парой байт

защёлкивается в триггер ChkIPDaddr по 5-му от начала "IP\_Daddr" фронту RxClk,

а общий результат сравнения адресов, включающий в себя результат сравнения

2-го слова со 2-й парой байт, защёлкивается по 9-му от начала "IP\_Daddr"

\_/ RxClk (на 1 такт позже, чем заканчивается приём "IP\_Daddr"). Значит, к

моменту, когда CtProto.q[] становится =7, и логика "Машины состояний" должна

решить, переходить к стадии приёма IP-DataBlock-a или нет, готов только ре-

зультат 1-го сравнения.

Поэтому по условию CtProto.q[]=7, независимо от результата сравнения адре-

сов, по следующему \_/ RxClk триггер RxIP\_Daddr переключается в =0, а триггер

RxUDPheadr переключается в =1 (т.е. "машина состояний" перейдёт к приёму

UDP-хедера, который размещён в начале IP-DataBlock-a). Далее, даже если сов-

пала 1-я половина IP-адреса (что вполне вероятно в рамках одной подсети),

то 2-я половина IP-адреса в "чужом" пакете точно НЕ совпадёт со 2-й половиной

IP-АдресаЭтогоУстройства. Итоговый результат сравнения защёлкивается в триг-

гер ChkIPDaddr через 1 RxClk после начала приёма UDP-хедера, а на 2-м от нача-

ла UDP-хедера такте RxClk формируется строб DadrIP\_cmp, по которому логика

состояния "RxUDPheadr" использует значение ChkIPDaddr.q для выбора дальнейших

действий :

- Если 2-я половина НЕ совпала, то уже через 1 RxClk после начала приёма

UDP-хедера триггер ChkIPDaddr переключится в =0, в течение 2-го от начала

UDP-хедера такта RxClk будет выставлен сигнал RxStop=1, и по 2-му \_/ RxClk

триггер RxUDPheadr переключится в =0. Таким образом, "машина состояний"

вернётся в исходное состояние, а по сигналу RxStop (который добавлен по

"ИЛИ" на вход Ether.RESET) приём Ethernet-фрейма будет прерван, и модуль

ETHERNET тоже перейдёт в исходное состояние.

- Если 2-я половина совпала, то приём UDP-хедера продолжится, и затем будет

принят UDP-шный DataBlock, т.е. будет нормально принят весь Ethernet-фрейм.

--============================================================================

ПЕРЕДАЧА ETHERNET-посылок

------------------------------

Обмен данными по протоколу UDP между Компом и Устройством, имеющим IP-адрес

192.168.AAA.DDD (где AAA - адрес подсети, DDD - адрес Устройства в подсети)

происходит следующим образом. Установление соединения по протоколу UDP начи-

нается с того, что Комп отправляет в подсеть ARP-запрос для того, чтобы узнать

MAC-адрес Устройства, имеющего IPaddr=192.168.AAA.DDD . Получив ARP-запрос,

Устройство должно запомнить MAC-адрес и IP-адрес Компа, и отправить ARP-ответ

согласно этим адресам. После этого и Комп и Устройство "знают" адреса друг

друга и =почти= готовы к обмену данными. Чтобы данные от Устройства стали

действительно доступны стандартному софту (например, MatLab-у) в Компе, нужно

ещё сообщить Устройству (текущее) значение UDP-порта, на который "открыт" дос-

туп из MatLab-овского скрипта, желающего обмениваться данными с Устройством.

Для этого нужно отправить из этого скрипта в Устройство UDP-пакет (хотя бы ми-

нимальной длины), а в этом пакете обязательно присутствуют поля "UDP-SourPort"

(UDP-порт отправителя, т.е. Компа) и "UDP-DestPort" (UDP-порт получателя, т.е.

Устройства). Получив этот пакет, Устройство должно запомнить значение

UDP-порта отправителя, и затем использовать его как UDP-порт получателя в от-

правляемых UDP-пакетах. Вот теперь MatLab-овский скрипт в Компе и Устройство

могут отправлять друг другу данные по протоколу UDP.

Во время нормальной работы даже при постоянном потоке UDP-пакетов от Устрой-

ства к Компу Комп (или Свитч) будет иногда посылать Устройству ARP-запрос.

Поэтому НЕОБХОДИМО заложить возможность правильной реакции на такую ситуацию.

А именно, если Компу (или Свитчу) удалось вклинить ARP-запрос между UDP-паке-

тами, отправляемыми Устройством, то нужно задержать начало отправки очередного

UDP-пакета и отправить ARP-ответ, после чего продолжить отправку UDP-пакетов.

Кроме того, ситуация существенно затрудняется ещё и тем, что в обычной ситу-

ации даже в минимальной "подсети", когда к компу подключена 1 линия с 1 Уст-

ройством, в эту линию комп время от времени отправляет кучу различных пакетов;

почти все эти пакеты являются broadcast-ными UDP-шными (отправлены, ессессно,

не в Устройство, а в другие IP-адреса). Любой такой пакет принимается и Уст-

ройство начинает его обрабатывать (т.к. пакет broadcast-ный) как "свой" до

"IP-Daddr" включительно, т.к. только по IP-Daddr видно, что пакет "чужой".

Необходимо обеспечить, чтобы принятые куски "чужих" пакетов не портили данные,

используемые в UDP-хедере отправляемых UDP-пакетов.

"ARP" :

-------

Когда в Устройство поступает ARP-запрос, после передачи поля "IP-Daddr"

итоговый результат сравнения поля "IP-Daddr" с IP-АдресомЭтогоУстройства пред-

ставлен значением ChkIPDaddr.q. На 2-м после окончания "IP-Daddr" такте RxClk

формируется строб DadrIP\_cmp, по которому логика обработки Ethernet-ных команд

использует значение ChkIPDaddr.q для выбора дальнейших действий (см. подраздел

"ETHERNET Commands proceeding unit") :

- Если IP-Daddr "чужой" <=> ChkIPDaddr=0, то ничего не делается.

- Если IP-Daddr "свой" <=> ChkIPDaddr=1, то триггер Ether\_ARPreply уста-

навливается в =1, и в конце приёма фрейма ARP-запроса по сигналу

Ether.RxEndFrame устанавливается запрос TxARPrepl\_Rq=1 на отправку ARP-от-

вета.

Т.к. работа с линией Ethernet производится в режиме Half-duplex, то во время

приёма ARP-запроса никаких других операций (передачи фреймов) с Ethernet-ом не

производилось => после приёма ARP-запроса линия Ethernet свободна, поэтому

сразу же, через минимально допустимую паузу после приёма ARP-запроса, должна

быть выполнена отправка ARP-ответа (т.к. это приоритетная операция, то даже

если до момента окончания паузы поступит также запрос на отправку UDP-пакета,

=первым= должен быть отправлен ARP-ответ). Поэтому при наличии TxARPrepl\_Rq=1

по окончании паузы сигналом EthBusyEnd устанавливается триггер InitARP =1 (а

установка триггера InitUDP заблокирована).

При переключении триггера InitARP в =1 вырабатывается импульс StartEthTx,

инициирующий цикл передачи Ethernet-фрейма модулем ETHERNET, а также на входы

модуля ETHERNET подаются значения Ether.TxLenTyp[]=H"0806"(тип="ARP") и

Ether.TxLength[]=46, и начальный адрес для вычитывания данных модулем ETHERNET

устанавливается =8(=8200). При инициализации Цикла\_Передачи схемка на тригге-

рах EthTxAdrIni2 и EthTxAdrInit вырабатывает дополнительный (1-й) строб для

загрузки начального адреса в счётчик адреса EtherAdrCt. Таким образом запус-

кается цикл передачи ARP-ответа.

Во время передачи ARP-ответа данные, передаваемые в DataBlock-е Ethernet-

фрейма, вычитываются из последовательных адресов addr=[8..30], поэтому никаких

установок адреса в процессе передачи не требуется. По окончании передачи

Ethernet-фрейма триггер Ether\_ARPreply сразу же сбрасывается в =0 сигналом

Ether.TxEndFrame, а после паузы, т.е. когда линия Ethernet =действительно=

доступна для следующей операции, сбрасывается в =0 и триггер EtherBusy.

Рис.4. Хронограмма сигналов во время инициализации передачи ARP-ответа.

--==============================================================================

--\*\*\*\*\*\*\*\*\*\*\*\* TRANSMISSION of ARP-reply \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

100MHz Clk100 \_\_/^^^\\_\_\_/^^^\\_\_\_/^^^\\_\_\_/^^^\\_\_\_/^^^\\_\_\_/^^^\\_\_\_/^^^\\_\_\_/^^^\\_\_\_/^^^\\_\_\_/^^^\\_\_\_/^^^\\_\_\_/^^^\\_\_\_/^^^\\_\_\_/^^^

EtherBusy ^^^^^^^^^^^\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

EthBusyEnd \_\_\_\_\_\_\_\_\_\_\_\_/^^^^^^^\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

InitARP \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\\_\_ ^^^^\\_\_ ^^^^\\_\_ ^^^^\\_\_

StartEthTx.q \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_StartEthTx.q

EthAdrInit \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

EthNextAdr[].D\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_////======08====================================================\\\\\_ ==\\\\\_ ==\\\\\_ ==\

EthAdrLd1 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/^^^^^^^^\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

EthAdrLd2 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/^^^^^^^^^^^^^^^^^^^^^^^\\_\_\_\_\_\_\_\_\_\_\_\_

EthNextAdr[].q \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/====08==============================

EthTxAdrIni2 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/^^^^^^^^^^^^^^^^^^^^^^^\\_\_\_\_

EthTxAdrInit \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/^^^^^^^\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

EthAdrCt\_en \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/^^^^^^^\\_\_\_\_\_\_\_\_\_\_\_\_

EthAdrCt.q[]->PortA \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/====08==============

enTransmit \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Ether.TX\_EN \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_///////////////////////////^^^^^^^

"IP" :

-------

Цикл отправки UDP-посылки может быть инициирован одним из следующих способов.

- Если в момент поступления запроса на отправку UDP-пакета модуль ETHERNET

был не "занят" (EtherBusy=0), то по сигналу TxUDP\_Rq сразу же устанавлива-

ется триггер InitUDP=1, и формируется импульс StartEthTx.

- Если в момент поступления запроса на отправку UDP-пакета модуль ETHERNET

был "занят" (EtherBusy=1), то сигнал TxUDP\_Rq вдвигается в "стек" ожидания

(на апрель 2019г "стек" глубиной 1 - это триггер TxUDPwait).

Когда ETHERNET закончит передачу предыдущего фрейма, то при [TxUDPwait=1 И

Ether\_ARPreply=0 (нет запроса на отправку ARP-ответа)] по окончании паузы

после отправки предыдущего фрейма сигналом EthBusyEnd установится триггер

InitUDP=1, и будет сформирован импульс StartEthTx.

Импульс StartEthTx инициирует цикл передачи Ethernet-фрейма модулем ETHERNET,

и одновременно с этим запускает в MainProject-e схему, которая обеспечивает

формирование UDP-посылки и подачу этих данных в ETHERNET.

Поле "IP-header'CRC" передаётся в 11-м и 12-м байтах IP-хедера, а значения

IP-Saddr и IP-Daddr, которые учтены в IP-headerCRC, передаются после него.

Поэтому значения IP-Saddr и IP-Daddr =необходимо= извлечь из ОЗУ и просуммиро-

вать заранее (до передачи IP-пакета); желательно также заранее просуммировать

значения 9-го и 10-го байтов (H"8011"), чтобы не пришлось "мгновенно" прибав-

лять их перед передачей поля "IP-header'CRC".

Контрольная сумма IP-header'CRC вычисляется во время передачи "DestAddr".

Процессом вычисления управляет микропрограммный автомат, основой которого яв-

ляется счётчик IPinitCt.

По сигналу InitUDP в счётчик IPinitCt загружается начальное значение =12.

Счёт (12->0) производится во время передачи поля "DestinationAddr". Перед

основным счётчиком IPinitCt сделан пред-делитель на 4 на триггерах IPinit1,

IPinit2 ; схемка построена так, что сигнал IPinit2 =1 в течение каждого 4-го

такта Clk100. Таким образом сигнал ([IPinit2, IPinit1]==3) формируется непос-

редственно на выходе триггера IPinit2, и его сразу (без промежуточной схемы

"2И") можно подавать на IPinitCt.cnt\_en (сэкономил аж 1 LCell).

Ниже дано краткое описание процесса вычисления IP-header'CRC (в виде Табли-

цы). Конкретные значения адресов (=EtherAdrCt.q[]) приведены для случая, когда

данные ARP-запроса/ARP-ответа расположены в ОЗУ начиная с addr=8 (и далее в

соответствии с описанием приёма ARP-запроса, см. выше); конкретные значения

накопленной суммы приведены для значений

IP-адрес Компа = 192.168.64.100 = H"C0A8 4064"

IP-адрес Этого\_Блока = 192.168.64. 10 = H"C0A8 400A" .

ТАБЛИЦА :

----------------------------------------------------------------------

IPinitCT EtherAdrCt Что просуммировано IPhdCRCadder

.q[] .q[] .result[]

12 15

11 15 IPSaddr(MSW) 0 + 49320 = 49320 = H"0 C0A8"

10 16 IPSaddr(MSW) + IPSaddr(LSW) + 16394 = 65714 = H"1 00B2"

09 17,18,19 --//----//--

08 20 --//----//-- + IPDaddr(MSW) + 49320 = 115034 = H"1 C15A"

07 21 . . . + IPDaddr(LSW) + 16484 = 131518 = H"2 01BE"

06 AdrInit 21

05 35 . . . + H"4000" + 16384 = 147902 = H"2 41BE"

04 36 . . . + H"8011" + 32785 = 180687 = H"2 C1Cf"

03 AdrInit 36

02 32 . . . + H"4500" + 17664 = 198351 = H"3 06CF"

01 32

00 32 и всё, остановка до передачи DataBlock-а.

После того как модуль ETHERNET передаст поля "Destin-nAddr" и "SourceAddr",

во время передачи "PacketLength/Type" он выдаёт в MainProj 1-й запрос IORq на

чтение 1-го слова данных, чтобы к моменту начала передачи 1-го байта данных

(MSByte 1-го слова данных) это слово уже было защёлкнуто в регистр Reg\_TxWord.

Далее процесс передачи данных из MainProject-а в Ethernet повторяется до окон-

чания передачи всего DataBlock-а.

Рис.4. Хронограмма сигналов во время передачи IP-пакета.

--==============================================================================

--\*\*\*\*\*\*\*\*\*\*\*\* TRANSMISSION of IP-packet \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

25MHz TxClk ^^^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^

CtTransm.q[] ===1=====X=====2=====X=====3=====\\_\_\_\_\_0\_\_\_\_\_/=====1=====X=====2=====X=====3=====X=====4=====X=====5=====X=====6=====X==7=

TxPkLenTyp ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

TxDataBlck \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

IORq (RdWr\_out=0)^^^^^^^^^^^^^^\\\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_///^^^^^^^^^^^^^^^^^^^^^^^^^^\\\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_///^^^^^^^^^^^^^^^^^^^^^^^^^^\\\\_\_\_\_

- -~8\*Clk100-->| |<------~8\*Clk100------>| |<------~8\*Clk100------>|

EthWord\_Wr \_\_\_\_\_\_\_\_\_\_\_\_\_\_///^^\\\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_///^^\\\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_///^^\\\\_\_\_\_

EtherAdrCt.q[]->PortA ==32==========XXX====================33=======================XXX=============================================XXX=

Ether.DataBus\_in[] ===DWord01=H"4500"==XXX\*\*\*\*\*\*X========== IP-Length=(UDPlength\_Reg[]+20) =====X== Num\_of\_UDP-Pkt=UDP\_PackCt.q[] =====

IPhdCRCadder.result[] =======================(IPSaddr+IPDaddr+H"4500")=================XX======(IPSaddr+IPDaddr+H"4500"+IPLength)======X

--\*\*\*\*\*\*\*\*\*\*\*\* TRANSMISSION of IP-packet -- continued \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

25MHz TxClk ^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_\_\_\_/^^^^^\\_\_

CtTransm.q[] ==17===X====18=====X====19=====X====20=====X====21=====X====22=====X====23=====X====24=====X====25=====X====26=====X====27==

IORq (RdWr\_out=0)^^^^^^^^^^\\\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_///^^^^^^^^^^^^^^^^^^^^^^^^^^\\\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_///^^^^^^^^^^^^^^^^^^^^^^^^^^\\\\_\_\_\_\_\_\_\_\_\_

- -~8\*Clk100-->| |<------~8\*Clk100------>| |<------~8\*Clk100------>|

EthWord\_Wr \_\_\_\_\_\_\_\_\_\_///^^\\\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_///^^\\\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_///^^\\\\_\_\_\_\_\_\_\_\_\_

EthAdrInit \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/^^^^^^^^^^^\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

EthAddrLd2 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_///^^^^^^^^^^^^\\\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

EtherAdrCt.q[]->PortA ====38====XXX====================39=======================XXX====================20=======================XXX=====

Ether.DataBus\_in[] = DWord07= ====XXX==========DWord08=IP-Saddr(LSWord)===========XXX==========DWord09=IP-Daddr(MSWord)===========XXX=

=IP-Saddr(MSWord)

Процесс передачи UDP-посылки состоит в том, что в то время когда модуль

ETHERNET передаёт DataBlock Ethernet-фрейма, в качестве данных подставляются

значения, составляющие IP-хедер и UDP-хедер, а затем - собственно данные,

которые нужно передать в DataBlock-e UDP-посылки. Ниже дано краткое описание

процесса передачи UDP-посылки (в виде Таблицы). Конкретные значения адресов

(=EtherAdrCt.q[]) приведены для случая, когда значения "по умолчанию" IP-хе-

дера расположены в ОЗУ начиная с addr=32.

ПЕРЕДАЧА UDP-пакета :

--------------------------

Ether.Tx EtherAdr Что читается :

Count[11..2] Ct.q[] Источник Значение

-------------+--------+---------------------------------------------------------

0(TxPkLenTyp) 32 EtherRAM.q\_a[] =H"4500"=17664

0(TxDataBlck) 33 IPlength.result[] = UDPlength+20

1 -//- 34 UDP\_PackCt.q[] = Num\_UDP-Pkt

2 -//- 35 EtherRAM.q\_a[] =H"4000"=

3 36 EtherRAM.q\_a[] =H"8011"=32785

4 37 IPhdCRCadder.result[] = CRC

5 38 EtherRAM.q\_a[] =IP-Saddr(MSWord)=ThisDev'IP(MSWord)

6 AdrInit 39 EtherRAM.q\_a[] =IP-Saddr(LSWord)=ThisDev'IP(LSWord)

7 20 EtherRAM.q\_a[] =IP-Daddr(MSWord)=Comp'IP(MSWord)

8 AdrInit 21 EtherRAM.q\_a[] =IP-Daddr(LSWord)=Comp'IP(LSWord)

9 58 EtherRAM.q\_a[] =UDP-SourPort =Comp'UDP-port

10 59 EtherRAM.q\_a[] =UDP-DestPort =Comp'UDP-port

11 AdrInit 60 UDPlength\_Reg[] =UDPlength

12 AdrInit 43 EtherRAM.q\_a[] =UDP-header'CRC ==0

13 62 EtherRAM.q\_a[] =UDP-DBlock DWord01

14 63 EtherRAM.q\_a[] =UDP-DBlock DWord02

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

--============================================================================

ТЕСТИРОВАНИЕ приёма/передачи данных

----------------------------------------

Наиболее детальное тестирование приёма/передачи данных включает в себя приём

пакетов (произвольных)\_данных различной длины, поступающих от Компьютера, и

последующую отправку пакетов с теми же данными обратно Компьютеру. В проект

(v.4i) заложены узлы, которые позволяют реализовать 2 режима тестирования :

\* ОДНОКРАТНАЯ отправка UDP-пакета по поступлению UDP-пакета данных от Компа.

\* МНОГОКРАТНАЯ отправка UDP-пакета, 1-й раз - по поступлению UDP-пакета данных

от Компа, а затем несколько раз (например, 11, т.е. всего 12) по 1 пакету

на каждую Линковую команду "StartNormal" (#193).

Узлы, инициирующие отправку UDP-пакетов, создаются в под-секции "2. Execution

of UDP command" секции "ETHERNET Commands proceeding unit".

Для ОДНОКРАТНОЙ отправки UDP-пакета, при поступлении от Компа UDP-пакета,

адресованного ЭтомуУстройству (опознавание "своих" пакетов описано в под-сек-

ции "IP" секции "ПРИЁМ ПАКЕТОВ различных типов"), через время задержжки 6-7мкс

после строба ChkIPDaddr формируется импульс TxUDP\_Rq\_UDP - запрос на отправку

UDP-пакета. Этот импульс, вместе с запросами от других источников, сводится

в TxUDP\_Rq - итоговый запрос на отправку UDP-пакета.

Для МНОГОКРАТНОЙ отправки UDP-пакетов, при поступлении от Компа UDP-пакета,

адресованного ЭтомуУстройству, триггер UDPtstSeries устанавливается в =1.

Этот сигнал разрешает отправку UDP-пакетов по командам "StartNormal" (#193),

поступающим по Линку. По окончанию отправки 12-го UDP-пакета (считая вместе

с 1-м пакетом, отправленным по получению исходного UDP-пакета от Компа) триг-

гер UDPtstSeries сбрасывается в =0, и передача пакетов по командам от Линка

прекращается ; передача следующей порции UDP-пакетов может быть инициирована

поступлением следующего UDP-пакета от Компа.

На данный момент, чтобы перевести Блок в режим тестирования, нужно в нижней

части под-секции "LINK Cycle proceeding units" закомментировать строку

"EtherTxRq\_Link = StartADC\_Link.q; " и раскомментировать следующую за ней

строку.

Во время нормальной работы, когда UDP-пакеты с данными оцифровки выпуливаются

из IPT-LXe с довольно большой частотой,

1) Какой должен быть минимальный интервал между пакетами ?

\*\*ОТВЕТ И.Логашенко: Точное значение пока неизвестно. Единицы микросекунд.

2) При такой ситуации, может ли у компа возникнуть желание послать ARP-запрос ?

И, если может, то как он вклинит ARP-запрос между часто поступающими UDP-па-

кетами ?

И, если может, то нужно ли приоритетно отправить ARP-ответ, приостановив на

это время отправку данных UDP-пакетами ?

\*\*ОТВЕТ И.Логашенко : даже при постоянном потоке UDP-пакетов от Устройства

нет гарантии, что комп НЕ будет иногда посылать этому Устройству ARP-запросы,

поэтому НЕОБХОДИМО заложить возможность правильной реакции на такую ситуацию.

А именно, если Компу (или Свитчу) удалось вклинить ARP-запрос между UDP-паке-

тами, отправляемыми Устройством, то нужно приостановить отправку очередного

UDP-пакета и отправить ARP-ответ, после чего продолжить отправку UDP-пакетов.

Yu: Ситуация немного облегчается тем, что

1) ARP-запрос может поступить только =между= UDP-пакетами, т.е. НЕ МОЖЕТ пос-

тупить =во время= передачи UDP-пакета. Поэтому не возникнет необходимости

ставить ARP-ответ "в очередь". Наоборот, если начался приём Ethernet-фрейма,

то запрос на отправку UDP-пакета ставится "в очередь" как минимум до конца

приёма пакета ; а если в принятом фрейме оказался ARP-запрос, то отправка

UDP-пакета откладывается до окончания отправки ARP-ответа.

2) Длительность Ethernet-фреймов ARP-запроса и ARP-ответа - немного меньше

6мкс, т.е. (приём ARP-запроса + пауза между приёмом ARP-запроса и отправкой

ARP-ответа + отправка ARP-ответа) = немного меньше 18мкс. Это меньше, чем

длительность отправки одного UDP-пакета, содержащего 256слов=512Байт (порция

данных от 1 канала в 1 событии).

Ваще-то, если внимательно проанализировать данные IP-хедера и UDP-хедера, то

видно, что в отправляемом UDP-пакете :

ПОСТОЯННЫЕ ПЕРЕМЕННЫЕ

DWord01=H"4500"

DWord02 - IP-Length

DWord03 - UDP-PktNum

DWord04=H"4000" (бывало H"0000")

DWord05=H"8011" (Type=UDP)

DWord06 - IP-header' CRC (соотв-ет IP-Length и UDP-PktNum)

DWord07 \\_ IP-Saddr (IP-шник моего Блока)

DWord08 / и то, может быть, надёжнее брать прямо из Константы

DWord09 \\_ IP-Daddr (IP-шник Компа)

DWord10 /

DWord11 - UDP-SourPort (UDP-port моего Блока)

DWord12 - UDP-DestPort (UDP-port Компа)

DWord13 - UDP-Length

DWord14 - IP-header' CRC (если UDP-SourPort=UDP-DestPort, то значение CRC

в отправляемом пакете будет то же, что и в принятом).

Видно, что из "установочного" UDP-пакета =необходимо= сохранять только значе-

ния UDP-SourPort и UDP-DestPort.

\* Значения DWord01, DWord04, DWord05, DWord07, DWord08, DWord11 можно ваще

прописАть в память изначально через mif-файл.

\* Значения [DWord09, DWord10] - IP-шник Компа - можно либо вычитывать из куска

памяти, в котором сохранён ARP-запрос, либо вовремя приёма ARP-запроса запи-

сать в "неприкосновенный" кусок памяти. Поскольку значение

END;