243-848-92 – Computer Project

Progress Report #1

Calvin Ouellet-Ference
A711643
Mr. Markou
Submitted on the 18th of February 2010

1. Objectives

The objectives of the first four weeks were to get the Ethernet controller fully functional. Which would complete the second block of the project (see figure 1in appendix for modular block diagram).

2. Ethernet controller

The original idea of using the Realtek RTL8019AS was dropped due to the 100-pin SMT and the various different voltages. It was then changed to the ENC424J600 but was too small to solder by hand so it was settled to the ENC28J60. From now one, eth0 will denote Ethernet Controller for simplicity.

The ENC28J60 does not have a parallel interface but a SPI one which resulted that I had to simulate an SPI interface with a PPI chip. Some functions to read and write to eth0 where built using the SET/RESET of the PPI as a clocking signal.

3. Circuits Schematics

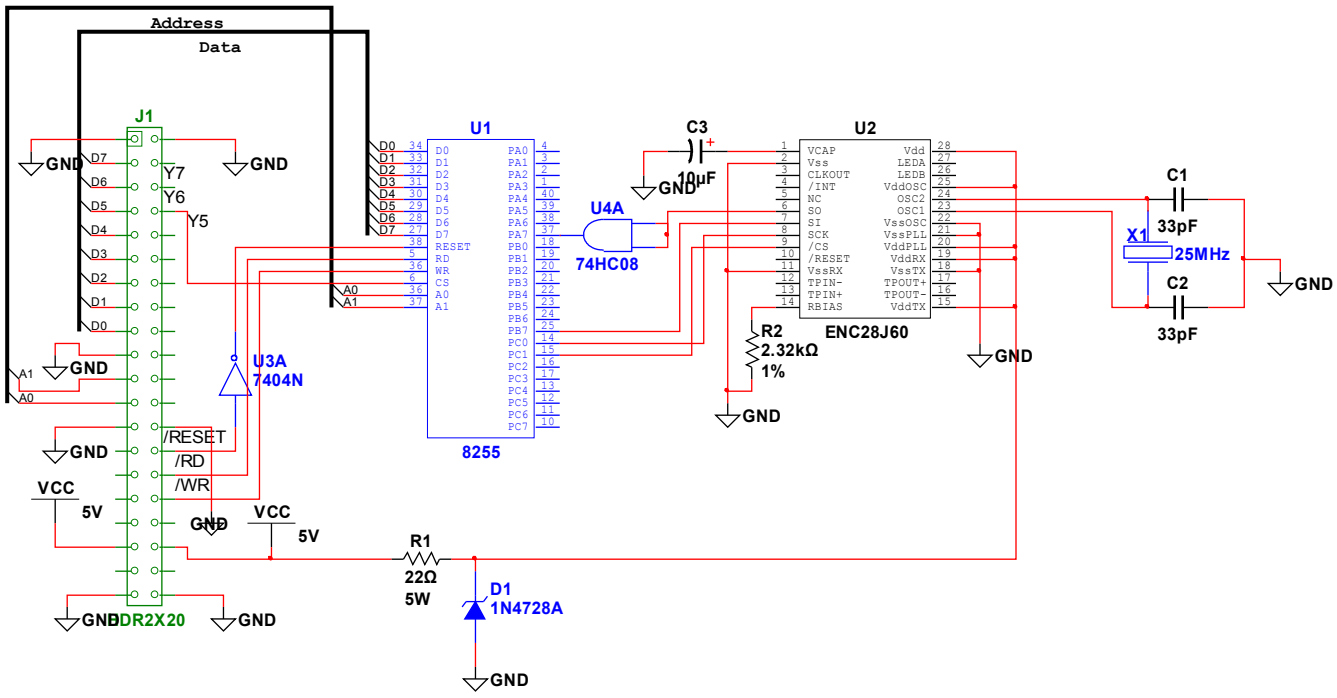


Figure 3.1 – eth0 with header

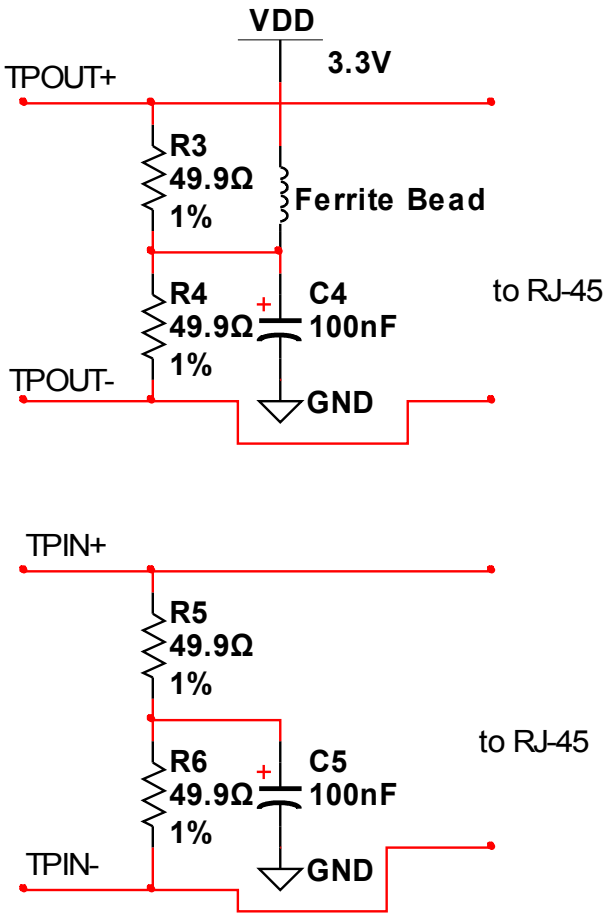
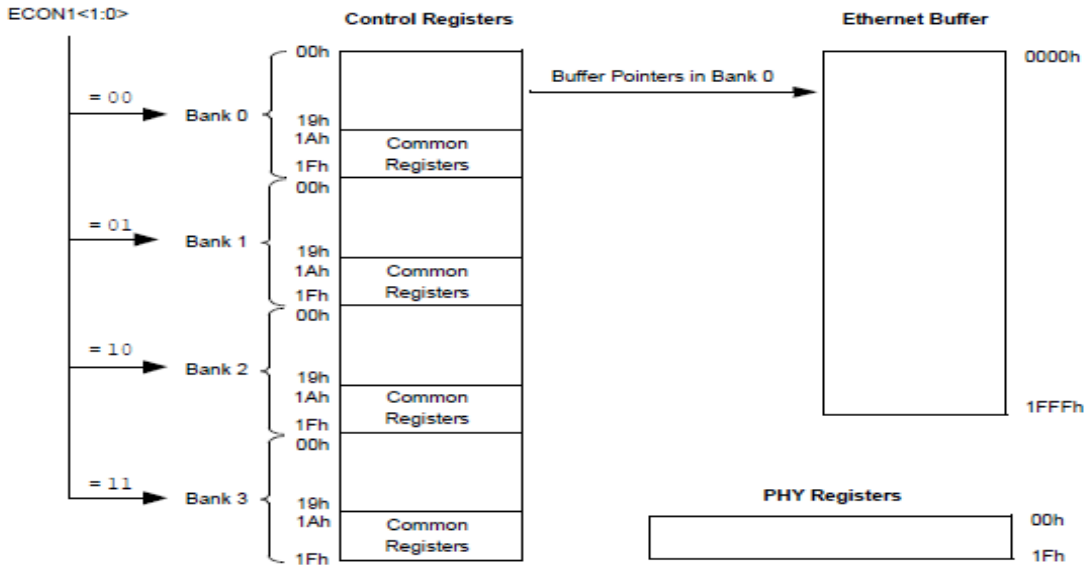


Figure 3.2 – RX/TX configuration from eth0

4. Code

To begin using eth0, first I need to initialize various registers. First the buffer size must be determined by programming the start and end of the receive buffer. In this application I split the 0x1FFF bytes of buffer into two, for 0x0FFF bytes for receive and transmission. The register to configure are the: ERXST and ERXND for the buffer size, then to configure the RX pointer that will determine which byte in the buffer is being read with the ERXRDPTL and ERXRDPTH buffers. Finally the ERXFCON needs to be configured to set the receive parameters.

Next is the configuration of the MAC registers. To keep it short, you must configure the MAC address and other various parameters including if it's half or full-duplex operation. I chose full-duplex as it is simpler to use. Finally the PHY registers need to be configured. These essentially are for the physical aspects of networking (e.g. the LED on the RJ45). These register are not accessed via the common register banks (shown below) but through some hidden registers.



Note: Memory areas are not shown to scale. The size of the control memory space has been scaled to show detail.

Figure 4.1 – ENC28J60 Memory Map

Above is the memory map of the Ethernet controller. The Common Registers are separated in four banks which are selectable through the ECON1 register. The Ethernet Buffer has a total of 2000 bytes of space available to it which I separated into two. Ranges from 0x0000 to 0x0FFF is for the receive buffer while range 0x1000 to 0x1FFF is for the transmit buffer. I'll probably never need more than that in both.

Bank 0 Address	Name	Bank 1 Address	Name	Bank 2 Address	Name	Bank 3 Address	Name
00h	ERDPTL	00h	EHT0	00h	MACON1	00h	MAADR5
01h	ERDPTH	01h	EHT1	01h	Reserved	01h	MAADR6
02h	EWRPTL	02h	EHT2	02h	MACON3	02h	MAADR3
03h	EWRPTH	03h	EHT3	03h	MACON4	03h	MAADR4
04h	ETXSTL	04h	EHT4	04h	MABBIPG	04h	MAADR1
05h	ETXSTH	05h	EHT5	05h	—	05h	MAADR2
06h	ETXNDL	06h	EHT6	06h	MAIPGL	06h	EBSTSD
07h	ETXNDH	07h	EHT7	07h	MAIPGH	07h	EBSTCON
08h	ERXSTL	08h	EPMM0	08h	MACLCON1	08h	EBSTCSL
09h	ERXSTH	09h	EPMM1	09h	MACLCON2	09h	EBSTCSH
0Ah	ERXNDL	0Ah	EPMM2	0Ah	MAMXFLL	0Ah	MISTAT
0Bh	ERXNDH	0Bh	EPMM3	0Bh	MAMXFLH	0Bh	—
0Ch	ERXRDPTL	0Ch	EPMM4	0Ch	Reserved	0Ch	—
0Dh	ERXRDPTH	0Dh	EPMM5	0Dh	Reserved	0Dh	—
0Eh	ERXWRPTL	0Eh	EPMM6	0Eh	Reserved	0Eh	—
0Fh	ERXWRPTH	0Fh	EPMM7	0Fh	—	0Fh	—
10h	EDMASTL	10h	EPMCSL	10h	Reserved	10h	—
11h	EDMASTH	11h	EPMCSH	11h	Reserved	11h	—
12h	EDMANDL	12h	—	12h	MICMD	12h	EREVID
13h	EDMANDH	13h	—	13h	—	13h	—
14h	EDMADSTL	14h	EPMOL	14h	MIREGADR	14h	—
15h	EDMADSTH	15h	EPMOH	15h	Reserved	15h	ECOCON
16h	EDMACSL	16h	Reserved	16h	MIWRL	16h	Reserved
17h	EDMACSH	17h	Reserved	17h	MIWRH	17h	EFLOCON
18h	—	18h	ERXFCON	18h	MIRDL	18h	EPAUSL
19h	—	19h	EPKTCNT	19h	MIRDH	19h	EPAUSH
1Ah	Reserved	1Ah	Reserved	1Ah	Reserved	1Ah	Reserved
1Bh	EIE	1Bh	EIE	1Bh	EIE	1Bh	EIE
1Ch	EIR	1Ch	EIR	1Ch	EIR	1Ch	EIR
1Dh	ESTAT	1Dh	ESTAT	1Dh	ESTAT	1Dh	ESTAT
1Eh	ECON2	1Eh	ECON2	1Eh	ECON2	1Eh	ECON2
1Fh	ECON1	1Fh	ECON1	1Fh	ECON1	1Fh	ECON1

Figure 4.2 – Common Registers

Above we can see the four banks and all the common registers, this is much more for a reference to the source code than anything relevant to the this report.

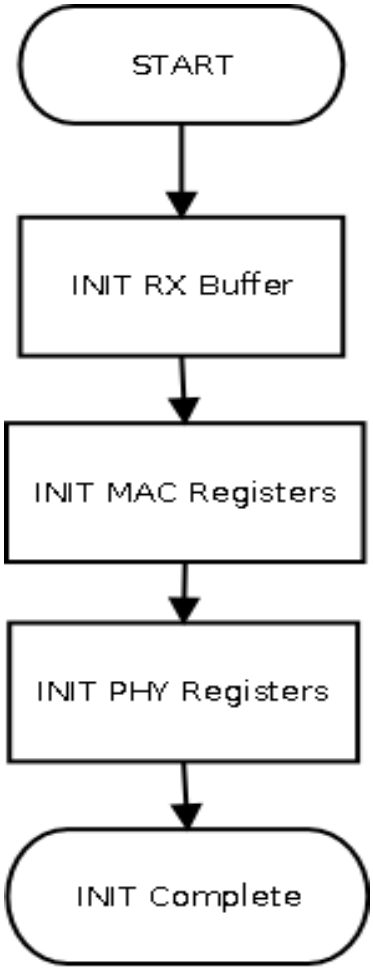


Figure 4.3 – Initialization Flow Chart

Above is a simple flow chart showing the steps in which the initialization is being done, below is the source:

```
.model tiny
.code
EXTRN        newline:NEAR, outbyte:NEAR, outword:NEAR, getc:NEAR, outstr:NEAR
; Eth0 preprocessor
ETH0    equ     0A00h   ;PPI2 + Eth_Controller
PORTA   equ     ETH0+0
PORTB   equ     ETH0+1
PORTC   equ     ETH0+2
PPICTL  equ     ETH0+3
SET             equ             00000001b
RESET           equ             00000000b
BANK0           equ             00010000b     ; eth0 Memory bank 0
BANK1           equ             00010001b
BANK2           equ             00010010b
BANK3           equ             00010011b     ; to 3
RDCR equ                00000000b     ; Read control register (Req, Arg.)
RDBMequ                 00111010b     ; Read buffer register
```

```
WCR             equ             01000000b       ; write control register (Req, Arg + Data Byte)
WBR             equ             01111010b       ; write buffer register (Req, Data Byte)
BFS             equ             10000000b       ; Bit field set (Req, Arg + Data Byte)
BFC             equ             10100000b       ; Bit field clear (Req, Arg + Data Byte)
SRC             equ             11111111b       ; Soft Reset

; To program any register, first the bank must be selected via ECON1

        org     0800h
main    proc
init:
        mov     al,10010000b    ;configuration word for the 8255
                                ;both group A and B = mode 0
                                ;port A = input
                                ;port B = output
                                ;port C = output
        mov     dx,PPICTL
        out     dx,al           ;send the configuration word
init_RX_Buffer:
; init_RX_Buffer requires to init: ERXSDT and ERXND first to determine the size of the Rx/Tx buffer.
                mov             ah,00000000b ; selects ECON1 and sets Bank 0 as active
                mov             al,1fh
                or              al,WCR                  ; adds the opcode
                call    outdata
; programming the start of Rx
                mov             al,08h          ; ERXSTL
                or              al,WCR
                mov             ah,00h
                call    outdata
                mov             al,09h          ; ERXSTH
                or              al,WCR
                mov             ah,00h
                call    outdata
; programming the end of Rx
                mov             al,0Ah          ; ERXNDL
                or              al,WCR
                mov             ah,0ffh
                call    outdata
                mov             al,0Bh          ; ERXNDH
                or              al,WCR
                mov             ah,0Fh
                call    outdata
; Now to program ERXRDPT
                mov             al,0Ch          ;ERXRDPTL
                or              al,WCR
                mov             ah,00h
                call    outdata
                mov             al,0Dh
                or              al,WCR
```

```
                mov             ah,00h
                call    outdata
; [Completed] INIT_RX_BUFFER
init_MAC:
; to init_MAC, first init MARXEN in MAXCON1
                mov             ah,00010010b ; selects ECON1 and sets Bank 2 as active
                mov             al,1fh
                or              al,WCR
                call    outdata
                mov             al,00h          ; MACON1 (Stands for MAc CONtrol 1)
                or              al,WCR
                mov             ah,0Dh
                call    outdata
                mov             al,02h          ; MACON3
                or              al,WCR
                mov             ah,0F5h
                call    outdata
                mov             al,03h          ; MACON3
                or              al,WCR
                mov             ah,40h
                call    outdata
                mov             al,04h          ; MABBIPG
                or              al,WCR
                mov             ah,15h
                call    outdata
                mov             al,06h          ; MAIPGL
                or              al,WCR
                mov             ah,12h
                call    outdata
; Now to prgoram the MAC Address with the following: 43:6F:66:66:65:65
                mov             ah,00010011b ; selects ECON1 and sets Bank 3 as active
                mov             al,1fh
                or              al,WCR
                call    outdata
                mov             al,04h          ; MAADR1 (MAc ADdRess 1)
                or              al,WCR
                mov             ah,43h
                call    outdata
                mov             al,05h          ; MAADR2
                or              al,WCR
                mov             ah,6Fh
                call    outdata
                mov             al,02h          ; MAADR3
                or              al,WCR
                mov             ah,66h
                call    outdata
                mov             al,03h          ; MAADR4
                or              al,WCR
                mov             ah,66h
```

```
                call    outdata
                mov             al,00h          ; MAADR5
                or              al,WCR
                mov             ah,65h
                call    outdata
                mov             al,01h          ; MAADR6
                or              al,WCR
                mov             ah,65h
                call    outdata
; [COMPLETED] init_MAC
init_PHY:
; Okay... here it gets complicated. I need to program the PHCON1 register... which is not part of the common registers
; to access the PHY registers, one must pass by the MII register in Bank 2. To program the PHY register, one must also
; Read from it first then Write to it... i wonder sometimes about these engineers...
                mov             ah,00010010b ; selects ECON1 and sets Bank 2 as active
                mov             ah,1fh
                or              al,WCR
                call    outdata
                mov             al,14h          ; MIREGADR
                or              al,WCR
                mov             ah,00h
                call    outdata
                mov             al,12h          ; MICMD
                or              al,WCR
                mov             ah,01h          ; enables read
                call    outdata
                mov             ah,00010011b ; selects ECON1 and sets Bank 3 as active
                mov             al,1fh
                or              al,WCR
                call    outdata
                ;call   debug
poll_init_phy:
                mov             al,0Ah
                or              al,RDCR
                call    indata
                and             al,01h
                cmp             al,01h
                je              poll_init_phy
                mov             al,12h                  ; MICMD
                or              al,WCR
                mov             ah,00h
                call    outdata                 ; set the read bit back to 0
                mov             al,1fh
                mov             ah,00010010b ; selects ECON1 and sets Bank 2 as active
                or              al,WCR
                call    outdata
                mov             al,19h          ; MIRDH
```

```
                or              al,RDCR
                call    indata
                mov             tmp,al
                mov             al,18h          ; MIRDL
                or              al,RDCR
                call    indata
                mov             tmp,ah
; Alright! Read completed!
                or              ah,00000001b
                mov             tmp,ah
                mov             ah,al
                mov             al,18h          ; MIRDL
                or              al,WCR
                call    outdata
                mov             ah,tmp
                mov             al,19h          ; MIRDH
                or              al,WCR
                call    outdata
                mov             ah,00010011b ; ECON1, Bank3
                mov             al,1fh
                or              al,WCR
                call    outdata
        ;       call    debug
poll_init_phy_2:
                mov             al,0Ah          ;MISTAT
                or              al,RDCR
                call    indata
                and             al,01h
                cmp             al,01h
                je              poll_init_phy_2
;[COMPLETED]  INIT_PHY
                mov             ah,00010100b                    ; selects ECON1 and sets Bank 0 as active
and Rx as Active
                mov             al,1fh
                or              al,WCR                  ; adds the opcode
                call    outdata
                ;call   debug
                ;call   txtst
here:   jmp here                                        ;standby


tmp             db 0
main    endp

udp_hdr db 26h, 17h, 26h, 17h, 1Ch, 00h, 00h
;source port(16)        : 0x2617
;dest. port (16)        : 0x2617
;length (16)            : 0x001C
;checksum (16)                  : 0x0000 (Default)
```

```
tcp_hdr db 26h, 17h, 26h, 17h, 00h, 00h, 00h, 00h
;source port (16)   : 0x2617
;dest. port(16)          : 0x2617
;sequence # (32)        : 0x0000 0x0000 (?)
;Ack # (32)                      :
;Data Offset (4)        :
;Reserve        (12)    :
;Window                 (16)    :
;Checksum     (16)    :
;Urgent ptr (16)        :
;Options        (24)    :
;Padding        (8)             :

ip_hdr db 45h, 00h, 00h, 14h, 01h, 40h, 00h, 06h, 00h, 00h, 0C0h, 0A8h, 00h, 0E1h, 0C0h, 0A8h, 00h,
0E1h, 0B0h, 00h ;last one is my data codes
;version (4)            : 0100 (4h)
;IHL (4)                        : 0101 (?) (5h)
;Type of Service (8): 0000 0000 (00h)
;Total Length  (16): 0000 0000 (00h), 0001 0100 (14h) (20d) don't need more, i'm sending single byte
codes... bad overhead
;ID      (8)                             : 0000 0001 (01h) (?)
;Flags (3)                      : 010 (2h) ; Don't fragment and Last fragment }
;Fragment offset(13): 0 0000 0000 0000 (00h)                                             } 0100 0000 0000
0000
;TTL    (8)                     : 1000 0000 (80h)
;Protocol (8)           : 0000 0110 (06h) (TCP=0x06 UDP=0x11)
;Header Checksum(16): 0000 0000, 0000 0000 (variable)
;Source address (32): 1100 0000, 1010 1000, 0000 0000, 1110 0001 (192.168.0.225)
;Destination AD(32) : 1100 0000, 1010 1000, 0000 0000, 1011 0000 (192.168.0.176)
;Options (24)           : No Need
;Padding (8)            : Most likely need some padding

debug proc              ; this is a debugging subroutine so that I can determine where the program is at
        push    ax      ; need to find a way so it tells me where the present debug output is in the
        push    di      ; program without having to count them
        lea             di,dbg
        call    outstr
        call    newline
        pop             di
        pop             ax
        ret
dbg             db "[DEBUG]",04
debug endp
```

The following are the subroutines for the PPI to simulate an SPI interface.

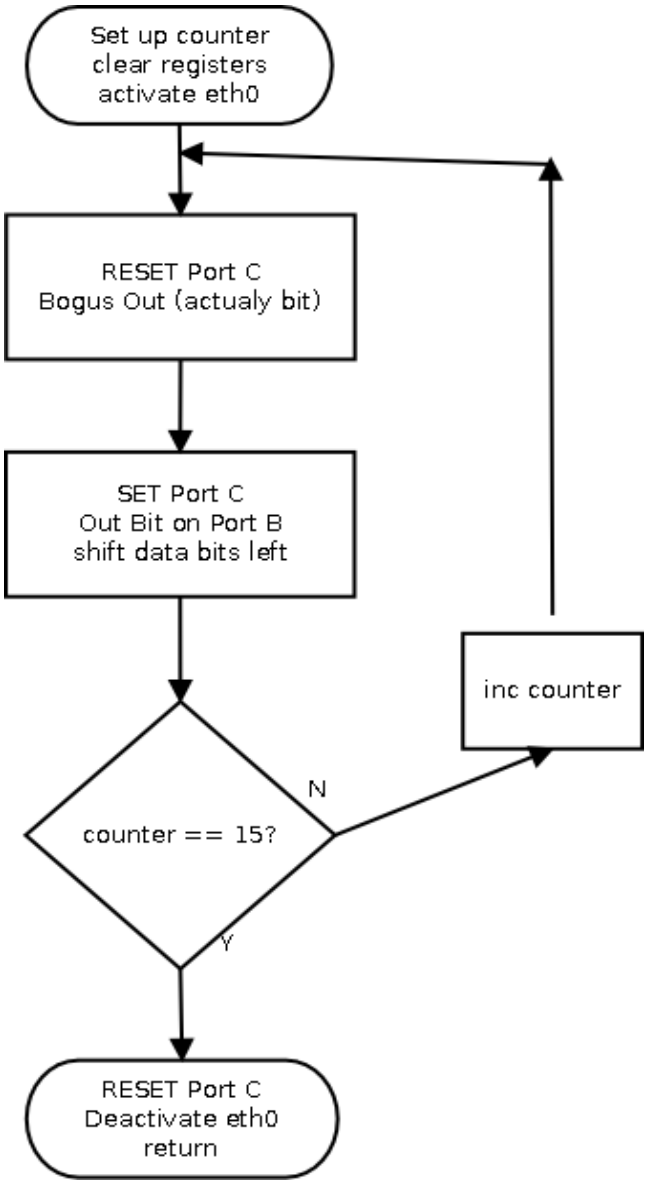


Figure 4.4 – outdata subroutine

```
outdata proc    ;[WORKING] also used for writing to registers
; Al.b7 Must have the first bit to send and ah must have the data to send
        push    bx                      ; need this for bogus
        mov             bx,0
        mov             cl,0            ; the bit counter
        mov             dx,PORTC
        push    ax
        mov             al,00h
        out             dx,al
```

```
        pop             ax
bogus:
        push    ax                      ; this is all bogus
        mov             dx,PPICTL       ; it is used to keep
        mov             al,RESET        ; a 50% duty cycle
        out     dx,al           ; on the SET/RESET
        pop             ax
        mov             dx,PORTB
        out             dx,al
        rol             bx,1
        cmp             bl,45
        je              outbit
        inc             bl
        jmp             outbit
outbit:
        push    ax
        mov             dx,PPICTL
        mov             al,SET
        out             dx,al           ; bit is set
        pop             ax
        mov             dx,PORTB
        out             dx,al           ; Out MSB (present)
        rol             ax,1            ; next bit to out
        cmp             cl,15
        je              shi
        inc             cl
        jmp             bogus           ; Not quite, but good for now
shi:
        mov             dx,PORTC
        mov             al,0f0h
        out             dx,al
        push    ax
        mov             al,RESET
        mov             dx,PPICTL
        out             dx,al
        pop             ax
        pop             bx
        ret
outdata endp
```

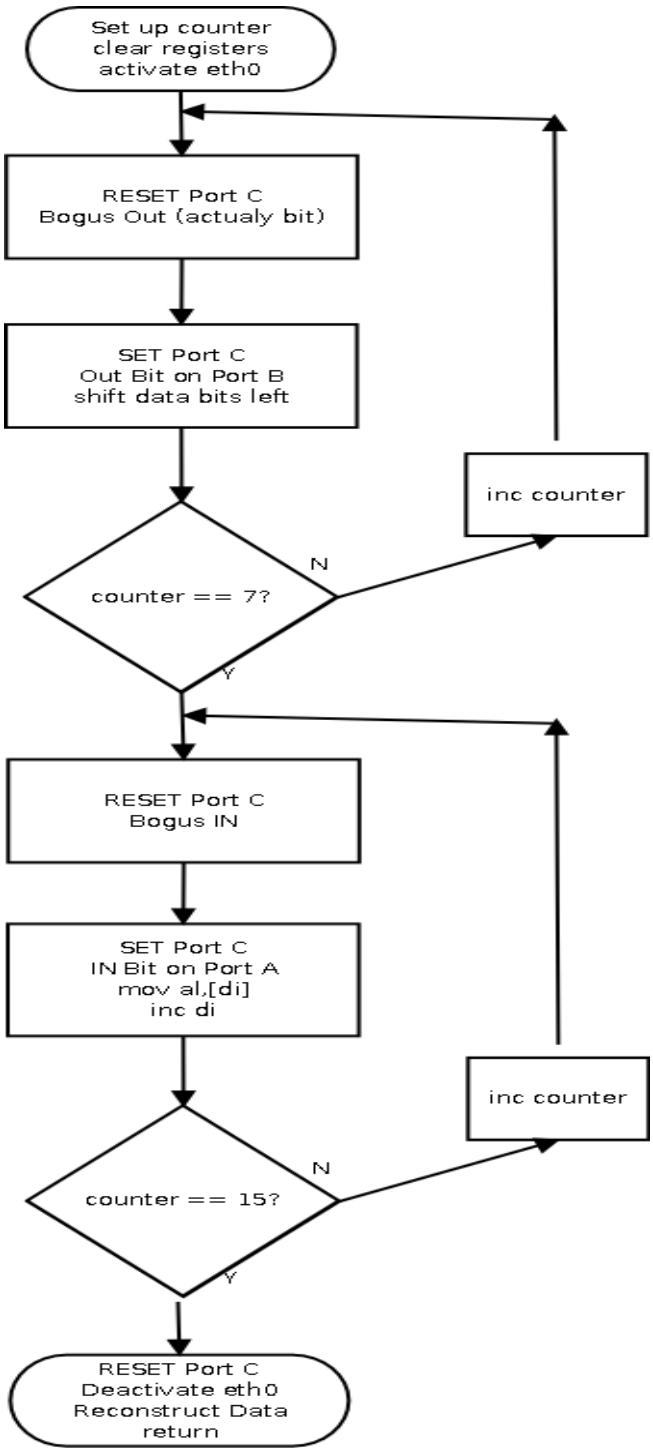


Figure 4.5 – indata subroutine

```
indata proc     ;[WORKING]
; Al.b7 Must have the first bit to send and ah must have the data to send
        push    bx                      ; need this for bogus
        mov             bx,0
        mov             cl,0            ; the bit counter
        lea             di,vals
        mov             dx,PORTC
        push    ax
        mov             al,00h
        out             dx,al
        pop             ax
bogus:
        push    ax                      ; this is all bogus
        mov             dx,PPICTL       ; it is used to keep
        mov             al,RESET        ; a 50% duty cycle
        out     dx,al           ; on the SET/RESET
        pop             ax
        mov             dx,PORTB
        out             dx,al
        rol             bx,1
        cmp             bl,45
        je              outbit
        inc             bl
        jmp             outbit
outbit:
        push    ax
        mov             dx,PPICTL
        mov             al,SET
        out             dx,al           ; bit is set
        pop             ax
        mov             dx,PORTB
        out             dx,al           ; Out MSB (present)
        rol             ax,1            ; next bit to out
        cmp             cl,7
        je              bogus2
        inc             cl
        jmp             bogus           ; Not quite, but good for now
bogus2:
        push    ax
        mov             dx,PPICTL
        mov             al,RESET
        out             dx,al
        pop             ax
        mov             dx,PORTA
        in              al,dx
        mov             [di],al
        inc             bx
        cmp             cl,45
```

```
        je      inbit
        inc     bl
        jmp     inbit
inbit:
        push    ax
        mov     dx,PPICTL
        mov     al,SET
        out     dx,al
        pop     ax
        mov     dx,PORTA
        in      al,dx
        mov     [di],al
        inc     di
        cmp     cl,15
        je      shi
        inc     cl
        jmp     bogus2
shi:
        mov     al,RESET
        mov     dx,PPICTL
        out     dx,al
        mov     dx,PORTC
        mov     al,0f0h
        out     dx,al
        lea     di,vals
        mov     cl,1

        mov     al,[di]
        and     al,80h
        mov     dl,al
        inc     di
        mov     al,[di]
        and     al,80h
        ror     al,cl
        inc     cl
        add     dl,al
        inc     di
        mov     al,[di]
        and     al,80h
        ror     al,cl
        inc     cl
        add     dl,al
        inc     di
        mov     al,[di]
        and     al,80h
        ror     al,cl
        inc     cl
        add     dl,al
        inc     di
```

```
        mov     al,[di]
        and     al,80h
        ror     al,cl
        inc     cl
        add     dl,al
        inc     di
        mov     al,[di]
        and     al,80h
        ror     al,cl
        inc     cl
        add     dl,al
        inc     di
        mov     al,[di]
        and     al,80h
        ror     al,cl
        inc     cl
        add     dl,al
        inc     di
        mov     al,[di]
        and     al,80h
        ror     al,cl
        add     dl,al
        mov     al,dl
        pop     bx
        ret
;debug:
;       mov     al,[di]
;       call    outbyte
;       inc     cl
;       cmp     cl,8
;       jb      debug
;       pop     ax
;       pop     bx
;       ret
vals db 0,0,0,0,0,0,0,0
indata endp
```

5. Conclusion

As far as the Ethernet controller side of the project goes, it is almost complete. Due to the milling machine and the acid bath being non-operational, I couldn't set up my RJ45 port due to it being non-standard footprint and will not fit in a socket. Once the PCB is built, testing for communication on the network will begin and the protocols will be written in the minimal system. The headers for IP and UDP were set up in the minimal system in the data section. Some early designs of the handshaking process have been designed (more or less) and will use the three-way handshake using UDP and not TCP. Because TCP is more complicated and is a connection oriented protocol. Some analysis of the ARP protocol was done and some strong considerations are being taken into implementing it into the minimal system as it appears not too complicated. Finally, some design changes on the sockets API were done due to the Windows API being utterly confusing and headache prone. I changed to the BSD API and will program under the latest version of PC-BSD.

Some ideas for determining water level in the coffee machine have been taken into consideration. Using a sliding potentiometer with an object which floats attached, using a pair of electrodes to determine water resistance or even a pressure transducer are being looked at. Also, a simple SPST relay will be used to start the coffee machine on and off which will be bought very soon.

Appendix

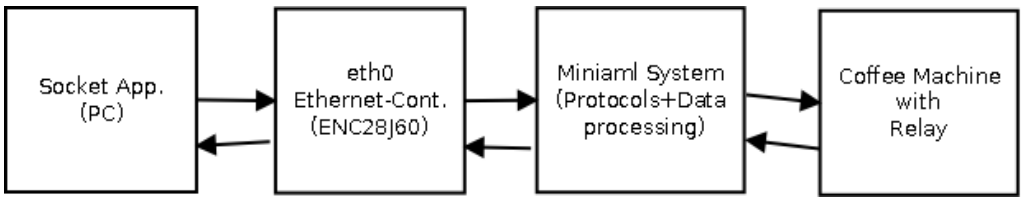


Figure 1