

CS4251 HW#2 Solutions

1. Below is the modified code.

```
#include <stdio.h>
#include <stdlib.h>

/* write out a value in hexadecimal */
void hex_write(FILE* f, unsigned long v) {
    fprintf(f, "%lX\n", v);
}

/* write out a value in binary */
void binary_write(FILE* f, long v, int bits) {
    int i;
    for(i = bits-1; i >= 0; i--) {
        if(v & (1<<i))
            putc('1', f);
        else
            putc('0', f);
    }
    putc('\n', f);
}

/* read a list of unsigned short values in binary;
 * assumes unsigned short is 16 bits */
int binary_read(FILE *f, unsigned short *buf, int max) {
    int ch;
    int bit;
    int len;
```

```

bit = 1;
len = 0;
buf[0] = 0;
while((ch = getc(f)) != EOF) {
    switch(ch) {
        case '1':
            buf[len] += 1<<(16 - bit); /* add bit */
            /* fall through to next case */
        case '0':
            if(++bit > 16) {
                bit = 1;
                if(++len > max) {
                    return len-1;
                }
                buf[len] = 0;
            }
            break;
        case ' ':
            /* ignore spaces */ break;
        case '\n':
            /* ignore newlines */ break;
        default:
            fprintf(stderr, "unknown bit: %c\n", ch);
            exit(1);
    }
}

return len;
}

/* the cksum code from the textbook, modified to print
 * intermediate output */
unsigned short cksum(unsigned short *addr, int count) {
    long sum = 0;

    printf("First loop:\n");
    while(count > 1) {
        sum += *addr++;
    }
}

```

```

        count -= 2;
        hex_write(stdout, sum);
        binary_write(stdout, sum, 32);
    }

    printf("Left-over byte:\n");
    if(count > 0) {
        sum += *addr;
        hex_write(stdout, sum);
        binary_write(stdout, sum, 32);
    }

    printf("Folding sum:\n");
    while(sum >>16) {
        sum = (sum & 0xffff) + (sum >> 16);
        hex_write(stdout, sum);
        binary_write(stdout, sum, 16);
    }

    return ~sum;
}

int main() {
    /* arbitrarily choose 128 as maximum length of
     * code to compute */
    unsigned short buf[128];
    int len;
    unsigned short sum;

    len = binary_read(stdin, buf, sizeof(buf)/sizeof(*buf));
    /* cksum expects number of bytes, not number
     * of shorts */
    sum = cksum(buf, len*2);
    printf("final sum:\n");
    binary_write(stdout, sum, 16);
    hex_write(stdout, sum);

    return 0;
}

```

```
}
```

Below is the input for the program:

```
11111111 11111111
11111111 00000000
11110000 11110000
11000000 11000000
```

Below is the output for the program:

```
First loop:
FFFF
00000000000000001111111111111111
1FEFF
00000000000000001111111011111111
2EFFF
00000000000000101110111111101111
3B0AF
00000000000000111011000010101111
Left-over byte:
Folding sum:
B0B2
1011000010110010
final sum:
0100111101001101
4F4D
```

2. a. Because CSMA/CD relies on sending nodes detecting collision, the act of the signal being placed onto the wire must last long enough so that nodes at the far end of the collision diameter will have detectable collisions. Because Gigabit Ethernet runs 10 times as fast as 100Mbit Ethernet, either the collision diameter must be reduced by a factor of 10 or the signal length must be increased by a factor of 10. To keep the collision diameter reasonable, they increased the minimum carrier time from 64 bytes to 512 bytes.

Note that they did not increase the minimum frame size. So if you send a 64 byte frame (the minimum), you pad the signal with 448 bytes.

Obviously if you have many of these frames, you're wasting significant bandwidth. To combat this, they allow packet "bursting" which allows several frames to be sent within a single carrier time slot.

Ref: article, page 6

b. The standard defines multimode fiber-optic, single-mode fiber optic, and copper links. Each link technology has a different maximum cable distance; they are, respectively: 550 meters, 3 kilometers, 25 meters.
Ref: article, page 10

c. Gigabit Ethernet does not provide Quality of Service. It relies on upper-level protocols like RSVP to handle this. It does, however, provide a method for "tagging" packets with a priority level that can be used by other protocols. Ref: article, pages 13-14

d. Ref: article, pages 2,7

Low Cost The per-port and bandwidth costs are much lower than those of FDDI.

Ease of Use Ethernet is much easier to deploy than a ring network. The same technology can be used as the backbone and to the desktop.

3. The slot reservation period must be longer to accommodate larger possible lengths. To represent 0 through k inclusive, we need $\log k + 1$ bits. With only a single slot available, we need only 1 bit. For choosing k , the use of the network needs to be considered. If k is too low, then you spend time doing reservation overhead when you could be transmitting actual data. That is, a station could be waiting for other stations to refuse reservations while it sits on data that could otherwise be transmitted. If k is too high, though, a single station can transmit while others are starved for access. Furthermore, if a fixed-length reservation encoding is used, then the size of the reservation bandwidth may overshadow the actual data being transmitted.

4. The timing diagram is shown in figure 1.

- P_x Poll (central to x)
- $Data_x$ Data (from x)
- G_x Go Ahead (from x to central)

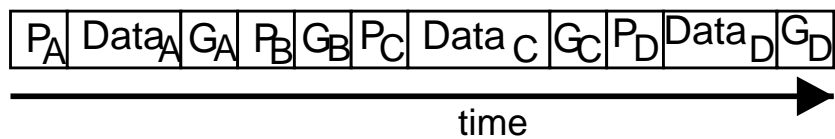


Figure 1: Timing Diagram for Problem 4