(15) **Question 1.** The inputs to a software function are \(x, y\) unsigned \(n\)-bit integers.

(5) Part a) An \(n\)-bit number is less or equal to \(2^n-1\), so the product of two \(n\)-bit numbers is less than \((2^n-1)\times(2^n-1)\), which is less than \((2^{2n}-1)\). So, the product \(x\times y\) will fit in a \(2n\)-bit number.

(5) Part b) If \(y = 1\), then the quotient \(x/1\) requires \(n\) bits. If \(y>1\), then \(x/y\) will be less than \(x\), so the quotient will always fit into an \(n\)-bit number.

(5) Part c) The input \(x\) ranges from 0 to \(2^n-1\), so the square root of \(x\) ranges from 0 to \(\sqrt{2^n-1}\). This will be less than \((2^{n/2}-1)\). So, the \(\sqrt{x}\) will fit in an \(n/2\)-bit number.

(5) **Question 2.**

C) CMOS logic needs charge whenever a digital line rises or falls. Current through the resistance of the wire will cause a voltage drop. The closer the capacitor is to the chip, the less voltage drop there will be on the \(V_{DD}\) pin.

(20) **Question 3.** Write a C program that scans a buffer and returns the mode of the data.

```c
char FindMode(char *pt){
    unsigned char i,mode;
    unsigned short Counts[256]={0}; // one count for each possible value
    unsigned short max;
    for(i=0; i<255; i++)
        Counts[(unsigned char)pt[i]]++; // count frequency of each
    max = 0;
    for(i=0; i<255; i++)// i is the value of data
        if(Counts[i]>=max){ // Counts is the number of that value
            max = Counts[i]; // find max count
            mode = i; // value at max is the mode
        }
    return (char)mode;
}
char FindMode( char *pt){
char answer;
    unsigned short maxtimes = 0,i;
    unsigned short Counts[256] = {0};
    for(i=0;i<1000;i++)
        Counts[*pt]++;
    if(Counts[(unsigned char)*pt] > maxtimes){
        maxtimes = Counts[(unsigned char)*pt];
        answer = *pt;
    }
    *pt++;
    return answer;
}
```

(35) **Question 4.** Put data out Port H clock into one 74HC374 using Port P rising edge CLK.

Part a) Show the hardware interface between the 9S12 and the three 74HC374 octal D flip flops.
Part b) Ritual makes all outputs zero.

```c
unsigned char Out0, Out1, Out2;  // output values
void Init(void){
    Out0 = Out1 = Out2 = 0;  // data=0
    DDRH = 0xFF;  // data outputs
    DDRP |= 0x07;  // clock outputs
    PTH = 0;  // data=0
    PTP &= ~0x07;  PTP |= 0x07;  // clock data into all three 374
    PTP &= ~0x07;
}
```

Part c) Write a C function that sets individual bits in the interface.

```c
void SetPort(unsigned char bit){unsigned char mask;
    mask = 1<<(bit&0x07);  // 1,2,4,8,16,32,64,128
    if(bit<8){
        Out0 |= mask;  // set bit
        PTH = Out0;  PTP_PTP0 = 1;  PTP_PTP0 = 0;  // latch
    }
    else if(bit<16){
        Out1 |= mask;  // set bit
        PTH = Out1;  PTP_PTP1 = 1;  PTP_PTP1 = 0;  // latch
    }
    else{
        Out2 |= mask;  // set bit
        PTH = Out2;  PTP_PTP2 = 1;  PTP_PTP2 = 0;  // latch
    }
}
```

Part d) Write a C function that clears individual bits in the interface.

```c
void ClrPort(unsigned char bit){unsigned char mask;
    mask = 1<<(bit&0x07);  // 1,2,4,8,16,32,64,128
    if(bit<8){
        Out0 &= ~mask;  // clear bit
        PTH = Out0;  PTP_PTP0 = 1;  PTP_PTP0 = 0;  // latch
    }
    else if(bit<16){
        Out1 &= ~mask;  // clear bit
        PTH = Out1;  PTP_PTP1 = 1;  PTP_PTP1 = 0;  // latch
    }
    else{
        Out2 &= ~mask;  // clear bit
        PTH = Out2;  PTP_PTP2 = 1;  PTP_PTP2 = 0;  // latch
    }
}
(10) **Question 5.** The data is clocked into 9S12 on the rising edge. Idle clock is high.

D) CPOL = 1; CPHA = 1

(15) **Question 6.** \( V_{out} = 5 - 2 \times V_{in} \). Use reference for constant. \( V_{ref} = 2.5V \). Use ground to make sum of gains equal to 1. \( V_g = 0 \). \( V_{out} = 2 \times V_{ref} - 2 \times V_{in} + V_g \). \( R_f/R_0 = 2 \), \( R_f/R_1 = 1 \), \( R_f/R_2 = 2 \).