Jonathan W. Valvano April 5, 2000, 11:00am-11:50am

(20) Question 1.
9600, 30 ft, half duplex, very high. Choose RS485 because it is a half-duplex protocol with good noise rejection.
You must use RS485 here, because it is the only half-duplex protocol listed (single wire OC was not a choice)

9600, 30 ft, full duplex, very high. Choose RS422 because it is a full-duplex protocol with good noise rejection.
You could use RS232, but it doesn't have as much noise rejection as RS422.
It depends on the magnitude of the noise, but digital logic probably would not work.

9600, 3000 ft, full duplex, low. Choose RS422 because it will work up to 4000 ft.
You must use RS422, because it is the only protocol that works at 3000 ft.

500,000, 30 ft, full duplex, medium. Choose RS422 because it can operate this fast and has good noise rejection.
Digital logic can run at 500,000 bps, but not in the presence of noise.

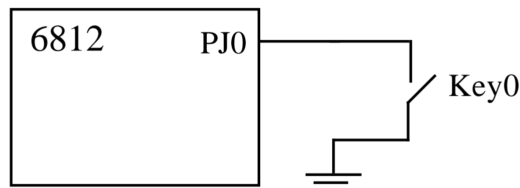
500,000, 3 in, full duplex, low. Choose digital because it is cheapest.
RS422 or RS232 could work, but they are more expensive than a simple set of wires.
3 inches means the two computers are on the same PC board or at least in the same box.

(20) Question 2. No changes to `OutChar()` because the hardware automatically generates parity on output, and no errors can occur when transmitting a frame.

```
void InitSCI(void){
    SC0BDH=0;  // 38400 BR=13 (MCLK=8MHz) clock
    SC0BDL=13; // BR=MCLK/(16*BaudRate)
    SC0CR1=0x12; // activate even parity (0x13 for odd parity)
    ParErr=0;
/* bit value meaning
    7   0    LOOPS, no looping, normal
    6   0    WOMS, normal high/low outputs
    5   0    RSRC, not applicable with LOOPS=0
    4   1    M, 1 start, 9 data, 1 stop
    3   0    WAKE, wake by idle (not applicable)
    2   0    ILT, short idle time (not applicable)
    1   1    PE, activate parity
    0   0    PT, even parity (1 for odd) */
    SC0CR2=0x0C; }
char InChar(void){
   while ((SC0SR1 & RDRF) == 0){};
   if(SC0SR1 & 0x01) ParErr++; // count number of parity errors
   return(SC0DRL);}
```

(60) Question 3. Recording songs
(10) Part a) Use Port J because it has internal pull-up resistors, which can be enabled.

(10) Part b) All 8 keys are separately connected, because we need to detect any combination of 0 to all 8 keys simultaneously pressed. Port J internal pull-up resistors are used, giving a zero if pressed and a one if released.



(5) Part c)

```
unsigned short n;       // index into Song[200], ranges from 0 to 200
// n is also the number of events that have been recorded, 200 means full
// Song[n] specifies where to store next event
```

```
unsigned short time;    // current time in 10 ms resolution
// varies from 0 to 65535
// 12345 means 123.45 sec
unsigned char previous; // value of Port J at the time of the last interrupt
```

(15) Part d) A simple approach is to use periodic polling. KeyWakeUp/OC could have been used to detect/debounce, but you still would have needed a second OC to maintain the 10 ms time clock. A 10 ms periodic polling rate was chosen because it is longer than the bounce, and matches the desired time resolution.

```
#define OC3 0x08
void ritual(void){
   asm{" sei"};       // make atomic
   DDRJ = 0;          // PortJ are all inputs
   PUPSJ=0xFF;        // pull-up
   PULEJ=0xFF;        // activate pull-up
   TSCR = 0x80;       // enable timer
   TMSK2 = 0x33;      // 1us TCNT
   TIOS |= OC3;       // activate output compare 3
   TMSK1 |=OC3;       // arm output compare 3
   TFLG1 =OC3;        // clear C3F
   TC3 = TCNT+10000;  // first interrupt at 10 ms
   previous=0xFF;     // assume initially no keys are touched
   n=0;               // index, Song[n] is where to store next event
   time=0;            // 0 sec
   asm{" cli"};       // enable
}
```

(15) Part e) The OC3 ISR records the song. Notice the exclusive or is used to detect changes. Notice Port J is read only once per interrupt. If you read it more than once, it might change in between reads.

```
#pragma interrupt_handler RealTimeClock()
void RealTimeClock(void){ int key; // varies from 0 to 7
unsigned char current;  // value of Port J now
unsigned char change;   // 1 in each bit that has changed since last interrupt
   TC3=TC3+10000;        // 10ms interval
   TFLG1 =OC3;           // acknowledge interrupt, clear C3F
   ++time;               // current time in 10 ms units
   current=PORTJ;        // the current value (0 means touch, 1 means release)
   change=current^previous; // zero if no change
   previous=current;     // previous is the value at the time of the last interrupt
   if(change){               // skip checks if no changes
      for(key=0;((key<8)&&(n<200));key++){
         if(change&0x01){               // this bit changed?
            Song[n].KeyCode=key;        // which bit has changed
            Song[n].What=current&0x01;  // 0 for touch, 1 for release
            Song[n].When=time;          // time of this event
            n++;                        // number of events
            change = change>>1;         // check bits 0,1,2,3,4,5,6,7
            current=current>>1;         // check bits 0,1,2,3,4,5,6,7
         }
      }
   }
   if((time==65535U)||(n==200))
        TMSK1 &=~OC3;   // disarm after 655.35 sec or 200 points
}
```

(5) Part f) The OC3 interrupt vector is placed at $FFE8

```
#pragma abs_address:0xffe8
void (*vector[])()={RealTimeClock};
#pragma end_abs_address
```