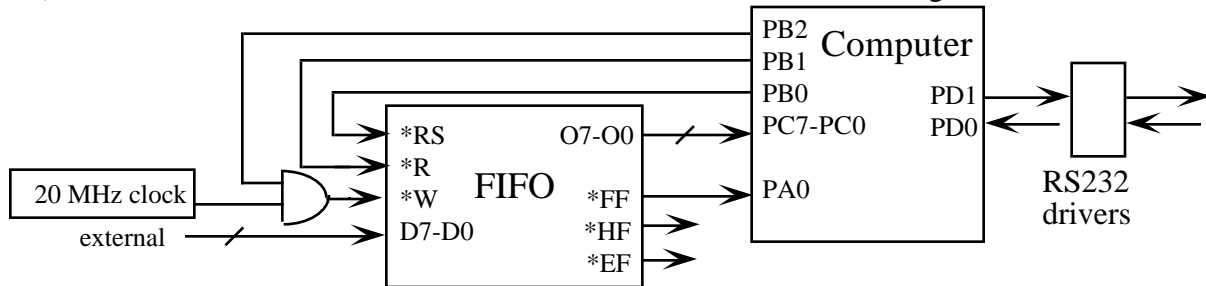


(35) Question 1.

Part a) Reset the fifo with PB0=0, read with PB1=0 and fill when PB2 is high. PA0 is 0 when full.



Part b) The system is dedicated (one task to do at a time, nothing else to do) so gadfly will work, plus the gadfly implementation will be faster than an interrupt solution.

```
void init(void) {
    BAUD=0x30;        /* 9600 baud */
    SCCR1=0x00;      /* 1start 8data, no par, 1stop */
    SCCR2=0x0C;     } /* tie=rie=0, te=re=1 */
#define RDRF 0x20
#define TDRE 0x80
void outsci(unsigned char letter) {
    while ((SCSR & TDRE) == 0); // Wait for transmit data register empty
    SCDR=letter; }
unsigned char insci(void) { // Gadfly wait for new input data
    while ((SCSR & RDRF) == 0);
    return(SCDR); }
void main(void){ unsigned char data; int i;
    init(); DDRC=0;
    while(1){
        PORTB=%010; // reset, no read, no fill
        insci(); // wait, discard value
        PORTB=%011 // no reset, no read, no fill
        PORTB=%111 // no reset, no read, fill
        while(PORTA&0x01); // wait for *FF=0
        PORTB=%011 // no reset, no read, no fill
        for(i=0; i<2048; i++){
            PORTB=%001 // no reset, read, no fill
            data=PORTC; // read while *R=0
            PORTB=%011 // no reset, no read, no fill
            outsci(data);}}}
```

(35) Question 2.

```
void wait(unsigned int time){
    TOC2=TCNT+time;
    TFLG1=0x40; // clear OC2F
    while((TFLG1&0x40)==0); // wait for time delay
}
void ReceiveMode(void){ // place the keyboard system in receive mode
asm(" sei"); // Make ritual atomic
    IRQvector=&STRAHan;
    DDRC=0x00; // Initially both are inputs
    PIOC=0x60;
// bit 6 STAI=1 arm,
// bit 5 CWOM=1 open collector,
// bit 4 HNDS=0 simple latched mode, STRB not used
// bit 3 OIN=X not applicable when HNDS=0
// bit 2 PLS=X STRB not used
// bit 1 EGA=0 for fall STRA means new data ready
// bit 0 INVB=X STRB not used
```

```

    flag=0;          // 1 byte FIFO empty
asm(" cli");}
#pragma interrupt_handler STRAHan()
void STRAHan (void){ unsigned int j, unsigned char dummy, frame;
    dummy=PIOC; dummy=PORTCL; // clear STAF
    frame=0;
    for(j=0;j<8;j++){
        while((PIOC&0x80)==0); // wait for high to low Clock
        frame=frame>>1; // bit0 first
        frame=frame|(PORTCL&0x80);} // shift in Data, clear STAF
    while((PIOC&0x80)==0); // wait for high to low Clock, parity
    dummy=PORTCL; // clear STAF
    while((PIOC&0x80)==0); // wait for high to low Clock, stop
    dummy=PORTCL; // clear STAF
    if(flag==0){ info=frame; flag=1;}
    DDRC=0x01; // Clock is output
    PORTC=0x00; // Clock is low
    wait(80);
    DDRC=0x00;}

void SendCommand(unsigned char command){ // transmit one command to the keyboard
    unsigned int j, unsigned char frame=command; unsigned char dummy, odd;
    PIOC=0x21; // disable STRA interrupts, set STAF on rise
    PORTC=0x81; // Clock and Data high
    DDRC=0x81; // Clock and Data floating
    while((PORTC&0x01)==0); // Waits for Clock to be high
// didn't use STAF because it may already have been high
    PORTC=0x80; // Set Clock low
    wait(800); // and wait for 400 μs
    PORTC=0x00; // Make Data low, while Clock still low
    DDRC=0x80; // Make Clock an input, while Data is still an output
    dummy=PIOC; dummy=PORTCL; // clear STAF
    odd=0x80; // calculate odd parity=number of one's in the 9 bits is odd
    for(j=0;j<8;j++){
        while((PIOC&0x80)==0); // wait for low to high Clock
        if(frame&0x01) {
            PORTC=0x80;
            odd^=0x80;} // flip parity bit for each "1" in the frame
        else
            PORTC=0x00;
        frame=frame>>1; } // bit0 first
    while((PIOC&0x80)==0); // wait for 9th low to high Clock
    PORTC=odd; // send odd parity
    DDRC=0x00; } // Leave both Clock and Data in input mode
(30) Question 3. This approach will work for all I/O ports with a direction register.
unsigned char LogicalDDRC; // simulated 6811 direction register
void PinInit(unsigned char direction){
    DDRC = 0x00; // 1 means output=0, 0 means input or output=1
    PORTC = 0x00; // we will toggle the DDRC to make output change
    LogicalDDRC = direction; // 1 means open collector output, 0 means input
    Pout(0xFF);} // (optional) output pins are already Hi Z
void Pout(unsigned char data){
// Pout does not affect DDRC bits of pins which are inputs
// for LogicalDDRC&data pins (high level=Hi Z outputs) make DDRC=0
    DDRC &= ~( LogicalDDRC&data );
// for LogicalDDRC&(~data) pins (low level=0 outputs) make DDRC=1
    DDRC |= ( LogicalDDRC&(~data) ); }
unsigned char Pin(void){ return PORTC;} // unchanged

```