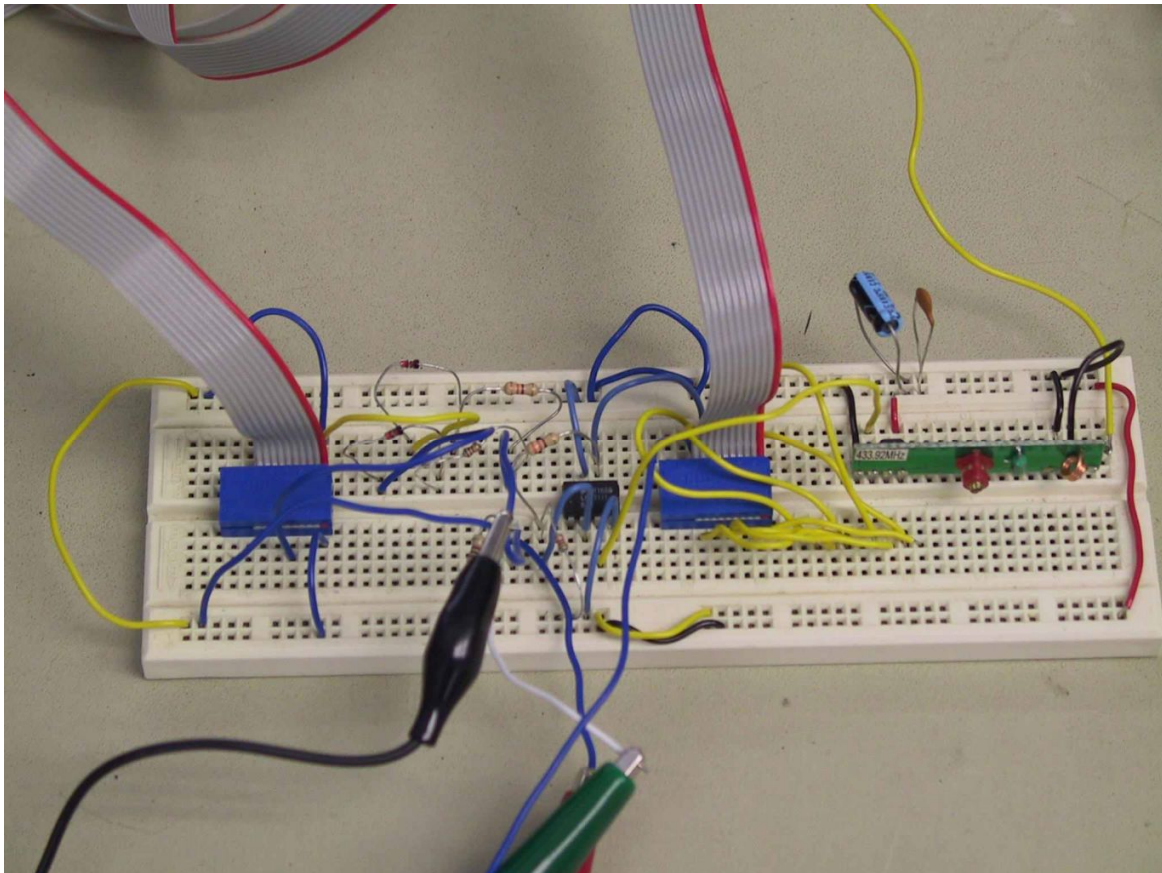


Wireless Drawing Device

Introduction:

For our final project, we want to build a wireless drawing device. The user uses a keypad or a mouse to draw on the TV through a wireless communication medium (RF - 433.92 MHz). The user should be able to move the drawing pointer to anywhere within the four border lines. One should also be able to draw or erase images on the TV screen.

In designing this project, we have two goals - a primary, which is realizable, and a secondary, which is feasible. Our realizable goal is to build a wireless keypad interface with the television. In this interface, we want the keypad to be able to draw and erase on the TV. When we accomplish our primary goal, we shall attempt to achieve our secondary goal: to build a wireless mouse having the same functionality as the keypad. Due to time constraints, we ended up only being able to complete the primary goal. We chose this project because we were inspired by wireless communication technology.



Before we begin our design, there are some issues that we need to be considered. There are two major parts to this wireless drawing device - a TV side and a keypad side. Therefore, there are two microcontrollers used: one for the TV side and one for the keypad side. The user interface (keypad side) of our drawing device consists of three buttons, left, right, and middle buttons, and a keypad or a mouse chassis. There is also a drawing pointer of size one pixel to ascertain where the drawing will occur on the TV.

Drawing Functions

There are three things that must be done in order to draw properly on the TV screen. The first is the drawing pointer movement. The pointer should be able to move up, down, left, right, and diagonally in all directions. The second thing is to draw on the TV screen. One needs to press the left button to draw wherever the drawing pointer is. Lastly, one needs to be able to erase the TV screen. One must press the right mouse button to erase wherever the mouse pointer is. If one presses the middle button, it will clear the TV screen and reset the drawing pointer to the center of the screen.

Wireless Communication

The connection is made via Universal Asynchronous Receiver and Transmitter (UART). We based our design on the Atmel Mega163 data sheet. On the keypad side, the microcontroller sends directional and button press signals to the transmitter via the TxD pin (PORTD.1), which is in turn received by the receiver on the TV side. These received signals go into the RxD pin (PORTD.0).

Also, the transmitter sends data at 433.92MHz. It has a maximum data rate to 4800bits/sec. We calculate the baud rate by using the formula given in the data sheet, $UBR = f_{ck}/16(UBR+1)$, where f_{ck} is the crystal clock frequency. This UBR equation can be found in table 27 in the data sheet. For a 8MHz crystal, $UBR = 103$ will give a baud rate of 4800.

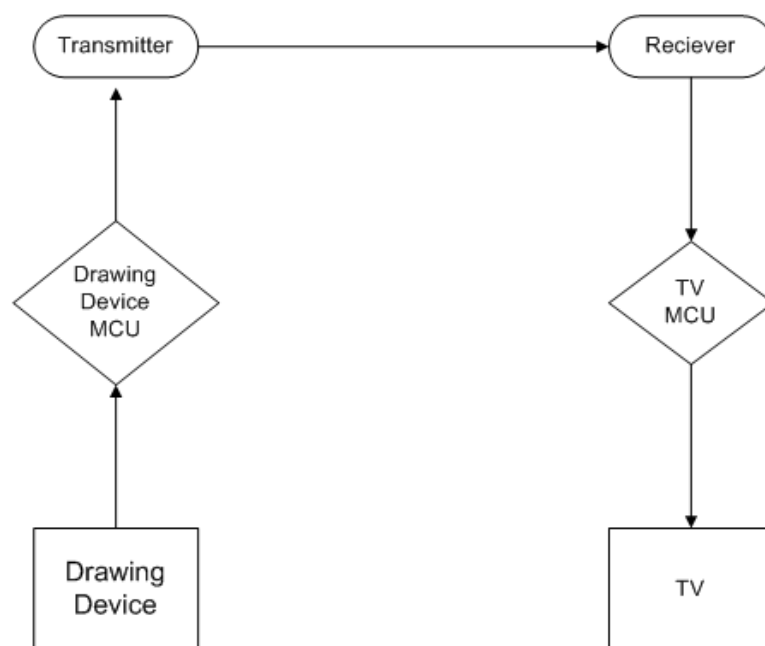
In order to get optimal transmission and reception, one needs to make the antenna approximately the size of one-quarter the wavelength of the transmitted signals. In our case, the wavelength is approximately 70 centimeters, and one-quarter of that is about 17 centimeters.

Basic Setup

Our drawing device consists of three functions. The first function is the movement the drawing device. For a keypad, one can press keys 1 through

9 for directions. Examples of these directions are 1 is move up and left, and 6 is move right. On the TV screen, the drawing pointer will move in the corresponding directions. The second function is to draw and erase on the TV depending whether the left or right button is pressed. Lastly, we need to encode the button presses and directional movements into a signal and send the encoded signal to the television side via the transmitter-receiver pair. On the TV side, we then need to decode the received signal in order to show the graphic on the TV screen.

The basic hardware connection is shown below:



Materials used:

- 2 Atmel Mega 163
- 1 keypad
- 3 buttons
- 1 transmitter: RCT-433-AS from Radiotronix
- 1 receiver: RCT-433-RP from Radiotronix
- 1 TV
- 4 AA batteries
- Two 0.1uF, and two 27pF capacitors
- Two 0.01uF, and two 4.7uF capacitors
- Two 330, two 1k, one 160, and one 75 Ohm resistors
- One 3-terminal Positive Regulator: LM340
- One operational amplifier: LMC7111

Hardware Design

The hardware design for this project is slightly tricky. The hardest and most time-consuming part of this design was getting all the hardware parts to communicate properly with each other. Constructing the keypad side microcontroller design, an off-board microcontroller, was interesting.

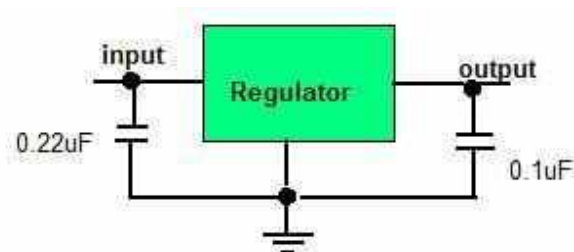
We followed the diagram given on the [page](#). We place one 27pF capacitor to pin 12 (XTAL2) and one to pin 13 (XTAL1) with a 8MHz crystal between those two pins. We place a 160 ohm resistor at pin 9 (reset') to pull the reset' to high to prevent constant resetting. Usually, reset' does not require a pullup, but with the RF transmissions, there is added RF distortion on the chip, which requires the pullup at reset'. After many tries with different resistor values, we found that a 160 Ohm resistor gave the best result at pulling up reset'. This does cause a decent amount of current to be sucked from the battery source, but when we tried using a 10k Ohm resistor, there was still constant resetting. To program the chip Pins 6 to 11 are connected to SPROG3. Between Vcc and Gnd, there is a 0.1uF capacitor.

On the TV side, there were a few connections needed between the TV MCU and the TV-side hardware. Here are the port/pin connections for the TV side:

PORT D	PORTD.0 to Receive data pin for TV Images
PORT A	PORTA.6 to video PORTA.5 to sync

NOTE: We place an operational amplifier between the receiver data pin and PORTD.0 to amplify the received signal. We amplified the signal from 3.8V to 4.6V.

On the keypad side

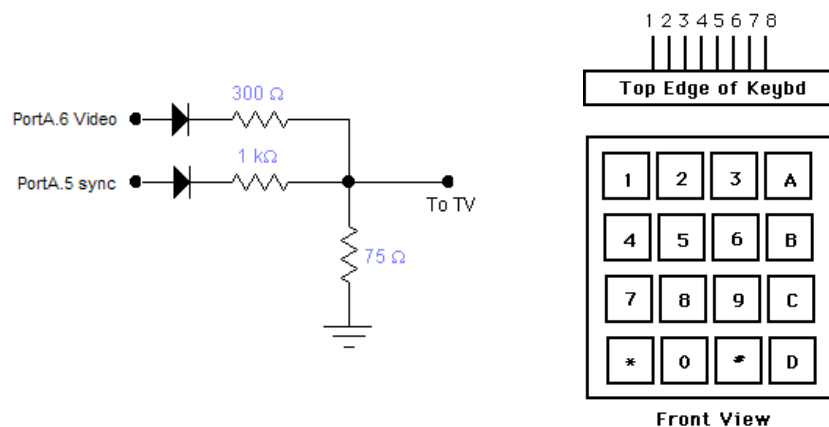


The input pin is connected to the positive side off the battery and the output pin is the 5V power supply of the board.

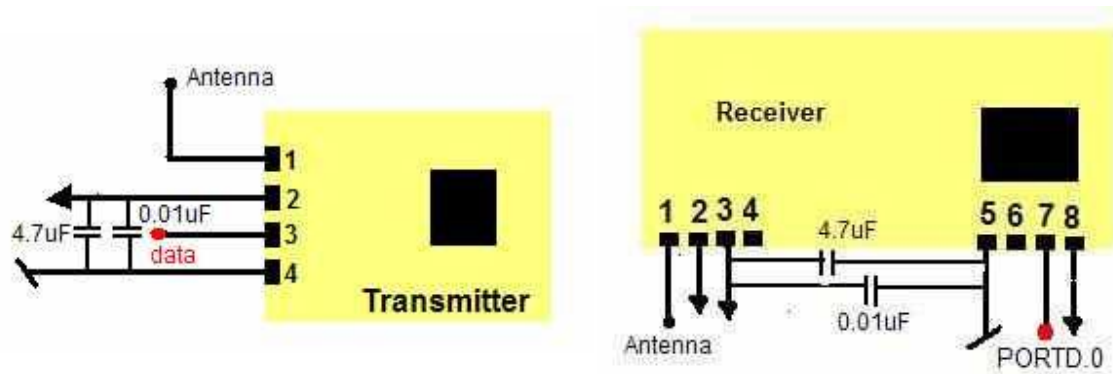
The keypad port/pin connections are listed below:

PORT B	PORTB.0 to button 1
	PORTB.1 to button 2
	PORTB.2 to button 3
PORT C	to Keypad
PORT D	PORTD.1 to transmitter data pin

Keypad and TV are connected as used in labs 5 and 3, respectively.



Transmitter and receiver connections to their respective MCU's are shown below:



Our software design can be divided into two different components.

Keypad Side

On the keypad side, we need to be able to detect whether the push buttons are pressed or not, and which one is pressed. These are taken care of in our "check_button" function by our state machine. The state diagram can be found in the . Additionally, we need to find the direction that the drawing pointer is moving. Below is a table that shows how each key press gets mapped into up, down, left, and right. This encoding scheme is done in our "encode_direct" function, which takes in four directions from the keypad. Our "get_direct" function determines which keypad directions are pressed. This state machine's state diagram can also be found in the appendix page.

Key Press	Up	Down	Right	Left
1	1	0	0	1
2	1	0	0	0
3	1	0	1	0
4	0	0	0	1
5	0	0	0	0
6	0	0	1	0
7	0	1	0	1
8	0	1	0	0
9	0	1	1	0

We use timer1 for the directional movement and timer2 for checking button presses. Both timers are run at 2ms.

Like we described on the High level design page, we use UART to transmit signals. We set the UART control register to Transmit Enable by setting `UCRSB = 0x08`. We also need to set the baud rate to 4800 by making `UBR = 103`. And we set `UDR = data_to_be_transferred` when we want to transmit data.

TV Side

On the TV side, we need to take care of the actual drawing. On reset or when the board first initializes, the television screen draws four lines, one on each edge of the screen, to signify the borders of the screen's drawing region. It also places the drawing pointer at the center of the drawing region. Using some code that we had implemented in laboratory three for the TV drawing, we got our television to display images. We want our drawing device to draw anywhere within the four border lines. We only use timer1 and run it at full speed on the TV side. Timer1 is interrupted after 509 timer ticks to make each frame exactly 1/60 of a second.

We use the "video_line" function to draw the border lines, "video_pt" function to draw any other images, and "video_set" function to determine whether a pixel is drawn on the screen or not. Each of these functions can be found in the laboratory three code and on the

During the vertical blanking (from line 231 to 262), We need to decode the received signal to determine what we want and where we want to draw on the television. We need to decode the drawing pointer direction and button presses for draw, erase, and clear screen. One can draw on the TV when the left button is pressed and erase when the right button is pressed. Pressing the middle button will clear the screen.

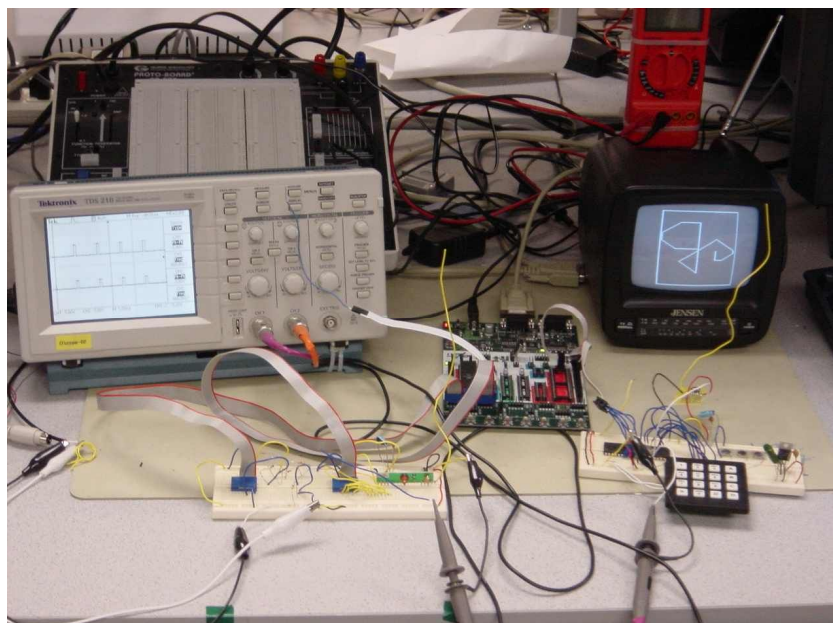
As described in the High level design page, we also use UART to receive signals. On the reception side, we set the UART control register to Receive Enable by setting `UCRSB = 0x10`. We also need to set the baud rate to 4800 by making `UBR = 103`. We set the `received_data = UDR`.

After much debugging and tweaking, we finally got our project to work as we specified. All the hardware testing and debugging took a great amount of time. Our wireless drawing device is able to transmit and receive data and draw anywhere on the TV. We did not get to implement it so that it will work like a mouse.

One thing that is worth mentioning here is that we had to make sure that the antennae for both the transmitter and receiver were fully extended in order to get best transmission and reception results. In fact, in testing our range, we could draw on the television from anywhere in the laboratory, which is about 15 meters. We were also able to walk out of the laboratory, all the way to the water fountain outside all the labs' area, and still draw on the television; that is about 20 meters. As long as there is not much RF interference, our range and reception are pretty good.

We could draw rather quickly - no measurable delay between movement direction and resulting television image. Our drawing mechanism was also rather accurate. When trying to draw, what we wanted to draw appeared as the image on the television.

Below is a picture of our entire setup working - transmission, reception, and images being drawn on the television. On the oscilloscope, one can see that the receiver is receiving what the transmitter transmitted. On the oscilloscope, the top line with the square waves is the transmitted signal. The bottom line is the received signal. One can see that they are identical with a slight delay of about 60us between transmission and reception.



After more than four weeks of designing, debugging, and tweaking, we have a fully functional wireless drawing device. We are able to use a wireless keypad to draw on a television screen. This project would have been much more intensive harder had we needed to code all the television timing and blasting to the television code ourselves. We are under our budget limit for the project. We spent a total of \$5.77 on the transmitter and receiver only. We got the regulator and op-amp from the lab

If we were to do this project again, we would have initially started with the keypad instead of trying to get the photosensors working. After getting the keypad wireless drawing to work, we would then implement the photosensors, so that the user can use our drawing device like a mouse to draw on the television.

Our project is a wireless design. We are not infringing on anyone's patents by building it, but had wireless mice not been around for so long, we may very well have broken some patent laws. Our wireless medium is RF with transmissions at about 433 MHz. Nothing in this project is too hazardous to one's health. Although, on occasion, the voltage regulator would get rather hot. When that occurred, we would merely disconnect the battery source to let it cool down. So, if this were a real marketable product, we would have to take care of that issue, as well as probably making it prettier to the eye.

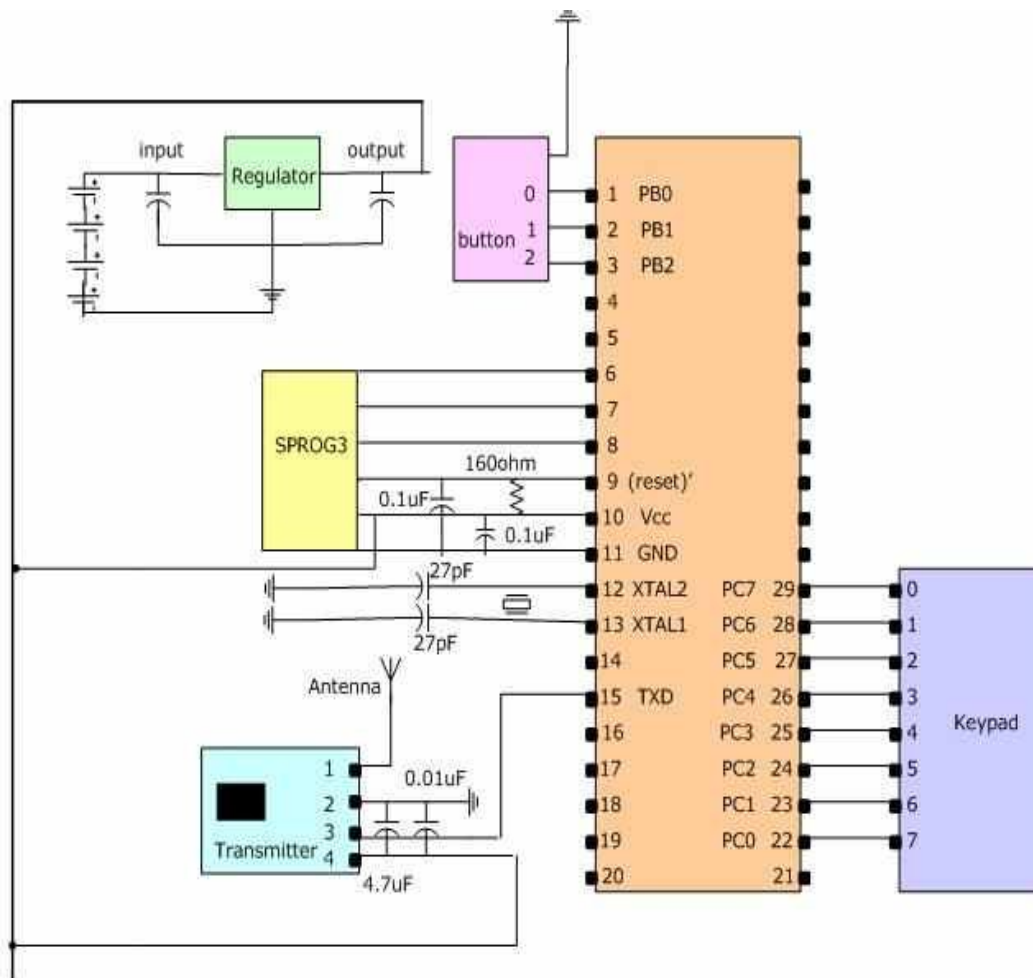
When we were working with our RF, there was usually at least one other group using the same frequency as we. Therefore, we needed to share the bandwidth. Obviously, sending all sorts of RF signals around the small lab room will cause interference in each other's transmissions, so we would always need to talk to the other to make sure that when we were running our transmissions, they would not and vice versa. Everyone that we talked with in the labs always seem agreeable to these circumstances.

Our original primary goal was to get the photosensors working and have the wireless drawing occur via the use of a mouse-like device; however, it took us far too much time to try to figure out how an optical photosensor worked. We searched every hardware site that we could find and asked the TA's and the Professor if they knew of a schematic on the device, but we could not find one. Since we had no schematic on this optical photosensor (it was a part laying around in the lab), by the time we got the photosensor running properly, we still needed to get the optical encoder and directional movements working. We truly wanted to make a wireless mouse - one that looked like a mouse, and not a keypad, for this drawing device. we believed that we could feasibly do so. However, the time constraints just got us. We had no working drawing device with about a week left, so we ported over to getting a wireless drawing keypad to work first, and then get back to trying

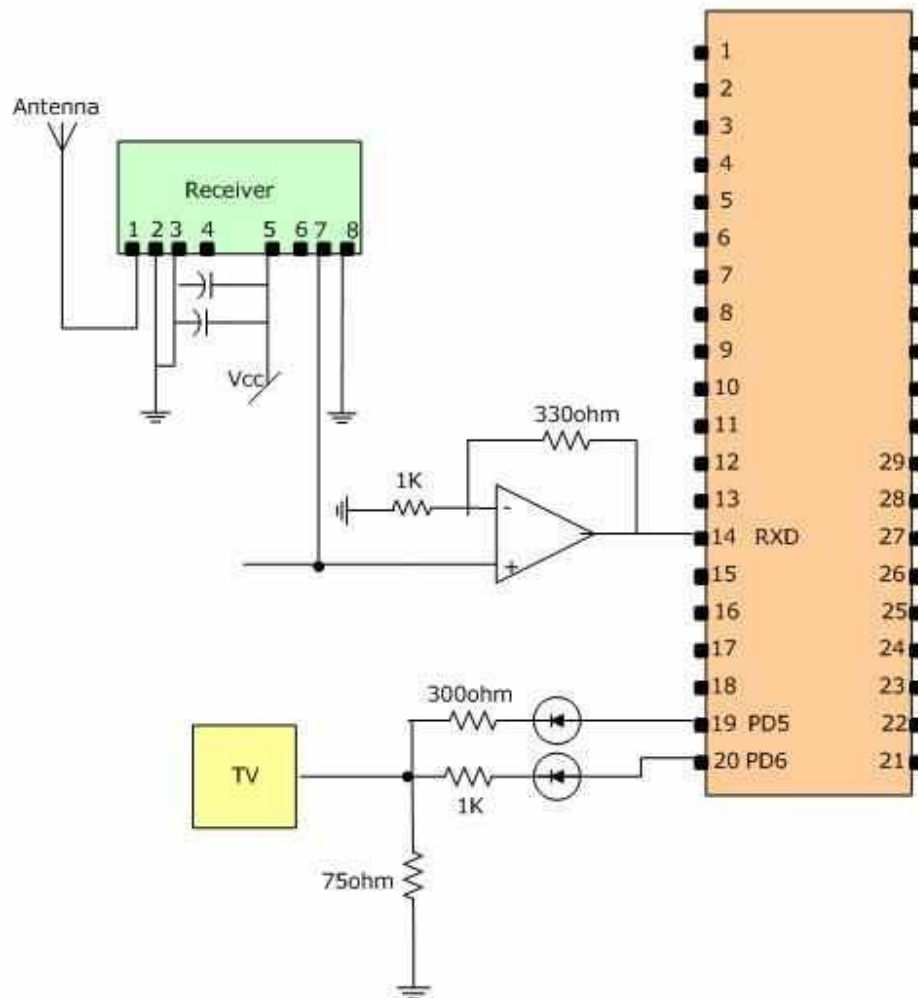
to get the optical photosensors working if there was still time remaining. So, we ended up having only a wireless keypad drawing device.

Throughout the lab sessions and extra hour sessions, the TA's, especially Sean and Chris, and the Professor were exceptionally helpful. There were times when some minute thing would just not work or times when we were trying to figure out a best design for our circuits, and we would be stuck trying to debug a tiny bug in our circuit or our program. It did not matter which of those three that we would ask, but each would always be quite able to help us.

Schematic for the keypad side:



Schematic for the TV side:



Keypad/mouse side Programmes:

```
// Emily Cheng, David Tow
// ECE 476, Final Project, Mouse-Side

#include <Megal63.h>
#include <stdio.h>
#include <stdlib.h>
#include <delay.h>

#define maxkeys 9

// set for timer1
#define t1 1          // get directional movement
#define t2 1          // check button presses

// Debounce State machine state names
#define NoPush        1
#define MaybePush     2
#define Pushed        3
#define MaybeNoPush   4

// define keypad state variables
#define Release 1
#define Debounce 2
#define Still_Press 3
#define Debounce_Release 4

unsigned char direct; // direction mouse is moving
unsigned char temp[6]; // temp var to put directional motion
unsigned char U, D, L, R;
unsigned char butL, butR, butM;
unsigned char PushState;

//keypad stuff
//key pad scan table
flash unsigned char keytbl[9]={0xee,0xde,0xbe,          // 1,2,3
                                0xed,0xdd,0xbd,          // 4,5,6
                                0xeb,0xdb,0xbb};          // 7,8,9

unsigned char KeyState;
unsigned char key_pressed;
unsigned char count, doink;
unsigned char butnum, maybe, key;

// send data
unsigned char transData;
unsigned char startTx;
unsigned char i;
unsigned char time1, time2;

void check_button(void);
void get_direct(void);
void keypad_call(void);
void encode_direct(void);
```

```

//*****
// timer1 compare ISR
interrupt [TIM1_COMPA] void t1_cmpA(void)
{
    if(time1 > 0) --time1;
    if(time2 > 0) --time2;
}

//*****
// start program
*****/
void main(void)
{
    //setup uART to transmit
    UCSRB = 0x08 ;           // uART to Tx enable on
    UBRR = 103 ;             // baud rate is 4800

    // initialization
    DDRB = 0x00;             // PORT B is for input -
pushbutton stuff
    PORTB = 0xff;

    direct = 0;
    U = D = L = R = butL = butR = butM = 0;
    transData = 0;
    PushState = NoPush;

    // setup timer1
    OCR1A = 16000;           // 16000 / 8e6 = 2ms
    TCCR1B = 9;              //full speed; clear-on-match
    TCCR1A = 0x00;           //turn off pwm and oc lines
    TIMSK = 0x10;           //enable interrupt T1 cmp

    // initialize the task timers
    time1 = t1;
    time2 = t2;

    KeyState = Release;
    key_pressed = 0;

    #asm ("sei"); // turn on interrupts

    startTx = 0xaa;
    // send 10101010, 3 times, to ensure proper gain of signal
    // aka sends 24 alternating 1's and 0's
    for(i=0;i<3;i++)
    {
        UDR = startTx; }

///////// end initialization

    while(1)
    {

```

```

        transData = 0;
        if(time1==0)    get_direct();
        if(time2==0)    check_button();
        transData = transData | butM | butL | butR | direct;

        // send transData to output port to send via transmitter
        // one bit at a time
        UDR = transData;
    } // end while
} // end main

////////////////////////////////////
// find the direction that the mouse is moving
void get_direct(void)
{
    time1 = t1;
    switch (KeyState)
    {
        case Release:
            if(butnum == 0)
            {
                encode_direct();
                KeyState = Release;
                keypad_call();
            }
            else
            {
                KeyState = Debounce;
                maybe = butnum;
                keypad_call();
            }
            break;

        case Debounce:
            if (butnum == maybe)
            {
                encode_direct();
                KeyState = Still_Press;
                keypad_call();
            }
            else
            {
                KeyState = Release;
                keypad_call();
            }
            doink = butnum;
            break;

        case Still_Press:
            // taking care of the hold button down and add 5 points
            // since call the debounce state machine every 3 ms,
            // this will add about 5 points every second if a key
    }
}

```

```

        if (count > 6)
        {
            encode_direct();
            count = 0;
        }

        // checking still-pressed condition
        if (butnum == doink)
        {
            count++;
            KeyState = Still_Press;
            keypad_call();
        }
        else
        {
            count++;
            KeyState = Debounce_Release;
            keypad_call();
        }
        break;

    case Debounce_Release:
        if (butnum == doink)
        {
            count++;
            KeyState = Still_Press;
            keypad_call();
        }
        else
        {
            count = 0;
            KeyState = Release;
            keypad_call();
        }
        break;
    }

} //get_direct

//////////
// encode direction
void encode_direct(void)
{
    direct = 0;
    // takes in four directions from external device somewhere
    // U, D, L, R = Up, Down, Left, Right
    switch (key_pressed)
    {
        case 1:
            U = 1;
            D = 0;
            L = 1;
            R = 0;
            break;
        case 2:
            U = 1;
            D = 0;

```



```

        L = 0;
        R = 0;
        break;
    case 3:
        U = 1;
        D = 0;
        L = 0;
        R = 1;
        break;
    case 4:
        U = 0;
        D = 0;
        L = 1;
        R = 0;
        break;
    case 5:
        U = 0;
        D = 0;
        L = 0;
        R = 0;
        break;
    case 6:
        U = 0;
        D = 0;
        L = 0;
        R = 1;
        break;
    case 7:
        U = 0;
        D = 1;
        L = 1;
        R = 0;
        break;
    case 8:
        U = 0;
        D = 1;
        L = 0;
        R = 0;
        break;
    case 9:
        U = 0;
        D = 1;
        L = 0;
        R = 1;
        break;
    default:
        U = 0;
        D = 0;
        L = 0;
        R = 0;
        break;
}

```

```

// encode directional movement
temp[0] = U << 2;
temp[1] = D << 3;
temp[2] = L << 1;

```

```

    temp[3] = R;
    // set mouse direction
    direct = direct | temp[0] | temp[1] | temp[2] | temp[3];
}

//////////
// check to see if pushbuttons are pressed
void check_button(void)
{
    time2 = t2;    // reset task timer
    switch (PushState)
    {
        case NoPush:
            if (~PINB == 0x01 || ~PINB == 0x02 || ~PINB == 0x04)
            {
                PushState=MaybePush;
            }
            else PushState=NoPush;
            break;
        case MaybePush:
            if (~PINB == 0x01 || ~PINB == 0x02 || ~PINB == 0x04)
            {
                PushState=Pushed;
            }
            else PushState=NoPush;
            break;
        case Pushed:
            if (~PINB == 0x01 || ~PINB == 0x02 || ~PINB == 0x04)
            {
                PushState=Pushed;
                if ((~PINB & 0x04) == 0x04)
                {
                    butR = 1;
                }
                if ((~PINB & 0x02) == 0x02)
                {
                    butM = 1;
                }
                if ((~PINB & 0x01) == 0x01)
                {
                    butL = 1;
                }
            }
            else PushState=MaybeNoPush;
            break;
        case MaybeNoPush:
            if (~PINB == 0x01 || ~PINB == 0x02 || ~PINB == 0x04)
            PushState=Pushed;
            else
            {
                PushState=NoPush;
                butR = 0;
                butL = 0;
                butM = 0;
            }
            break;
    }
}

```

```

        // encode mouse button presses
        butR = butR << 4;
        butL = butL << 5;
        butM = butM << 6;
    } // end check_button

////////////////////////////////////
// call to check what, if anything, was inputted on the keypad
void keypad_call(void)
{
    //get lower nibble
    DDRC = 0x0f;
    PORTC = 0xf0;
    delay_us(5);
    key = PINC;

    //get upper nibble
    DDRC = 0xf0;
    PORTC = 0x0f;
    delay_us(5);
    key = key | PINC;

    //find matching keycode in keytbl
    if (key != 0xff)
    {
        for (butnum=0; butnum<maxkeys; butnum++)
        {
            if (keytbl[butnum]==key) break;
        }
        if (butnum==maxkeys) butnum=0;
        else butnum++; //adjust by one to make range 1-9
    }
    else butnum=0;
    key_pressed = butnum;
} //keypad_call

```

TV side Program Code:

```

// David Tow and Yi Fan Cheng
// ECE 476, Final Project, TV-side

//video gen
//D.5 is sync:1000 ohm + diode to 75 ohm resistor
//D.6 is video:330 ohm + diode to 75 ohm resistor

#include <Mega163.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

//cycles = 63.625 * 8 Note NTSC is 63.55
//but this line duration makes each frame exactly 1/60 sec
//which is nice for keeping a realtime clock

```

```

#define lineTime 509

#define ScreenTop 30
#define ScreenBot 230
#define T0reload 256-60

#pragma regalloc-
//NOTE that v1 to v8 and i must be in registers!
register char v1 @4;
register char v2 @5;
register char v3 @6;
register char v4 @7;
register char v5 @8;
register char v6 @9;
register char v7 @10;
register char v8 @11;
register int i @12;
#pragma regalloc+

char syncON, syncOFF;
int LineCount;

char screen[800];

//Point plot lookup table
flash char pos[8]={0x80,0x40,0x20,0x10,0x08,0x04,0x02,0x01};

// Used for inputs for directional positioning
unsigned char up, down, left, right;          // input is the information
received from the mouse-side
unsigned char Lbutt, Rbutt, Mbutt;            // button presses
unsigned char x,y;
unsigned char x1,y1;
unsigned char pixelSet, pixFlag;

// used to receive data
unsigned char maskedRxd;
unsigned char input;

//=====
//This is the sync generator. It MUST be entered from
//sleep mode to get accurate timing of the sync pulses
//At 8 MHz, all of the sync logic fits in the 5 uSec sync
//pulse
interrupt [TIM1_COMPA] void t1_cmpA(void)
{
    //start the Horizontal sync pulse
    PORTA = syncON;
    //count timer 0 at 1/usec
    TCNT0=0;
    //update the curent scanline number
    LineCount ++ ;
    //begin inverted (Vertical) synch after line 247
    if (LineCount==248)
    {
        syncON = 0b00100000;
    }
}

```

```

        syncOFF = 0;
    }
    //back to regular sync after line 250
    if (LineCount==251)
    {
        syncON = 0;
        syncOFF = 0b001000000;
    }
    //start new frame after line 262
    if (LineCount==263)
    {
        LineCount = 1;
    }

    //end sync pulse
    PORTA = syncOFF;
}

//=====
//plot one point
//at x,y with color 1=white 0=black 2=invert
void video_pt(char x, char y, char c)
{
    //The following odd construction
    //sets/clears exactly one bit at the x,y location
/*
    i=((int)x>>3) + ((int)y<<3) ;
    if (c==1) screen[i] = screen[i] | 1<<(7-(x & 0x7));
    if (c==0)     screen[i] = screen[i] & ~(1<<(7-(x & 0x7)));
    if (c==2)     screen[i] = screen[i] ^ (1<<(7-(x & 0x7)));
*/

    //The following odd construction
    //sets/clears exactly one bit at the x,y location
    #asm
    ;i=(x>>3) + ((int)y<<3) ;
;   push r30
;   push r31
    push r16
    ldd r30,y+2          ;get x
    lsr r30
    lsr r30
    lsr r30              ;divide x by 8
    ldd r12,y+1          ;get y
    lsl r12              ;mult y by 8
    clr r13
    lsl r12
    rol r13
    lsl r12
    rol r13
    add r12, r30          ;add in x/8

    ;v2 = screen[i];    r5
    ;v3 = pos[x & 7];   r6
    ;v4 = c              r7
    ldi r30,low(_screen)

```

```

ldi r31,high(_screen)
add r30, r12
adc r31, r13
ld r5,Z           ;get screen byte
ldd r26,y+2       ;get x
ldi r27,0
andi r26,0x07     ;form x & 7
ldi r30,low(_pos*2)
ldi r31,high(_pos*2)
add r30,r26
adc r31,r27
lpm r6,Z
ld r16,y          ;get c

;if (c==1)        screen[i] = screen[i] | 1<<(7-(x & 0x7));
;if (c==0)        screen[i] = screen[i] & ~(1<<(7-(x & 0x7)));
;if (c==2)        screen[i] = screen[i] ^ (1<<(7-(x & 0x7)));
;if (v4==1) screen[i] = v2 | v3 ;
;if (v4==0) screen[i] = v2 & ~v3;
;if (v4==2) screen[i] = v2 ^ v3 ;
cpi r16,1
brne tst0
or r5,r6
tst0:
cpi r16,0
brne tst2
com r6
and r5,r6
tst2:
cpi r16,2
brne writescrn
eor r5,r6
writescrn:
ldi r30,low(_screen)
ldi r31,high(_screen)
add r30, r12
adc r31, r13
st Z, r5

pop r16
;      pop r31
;      pop r30
#endasm

```

```

}

```

```

//=====
//plot a line
//at x1,y1 to x2,y2 with color 1=white 0=black 2=invert
//NOTE: this function requires signed chars
//Code is from David Rodgers,
//"Procedural Elements of Computer Graphics",1985
void video_line(char x1, char y1, char x2, char y2, char c)
{
    int e;

```

```

signed char dx,dy,j, temp;
signed char s1,s2, xchange;
signed char x,y;

x = x1;
y = y1;
dx = cabs(x2-x1);
dy = cabs(y2-y1);
s1 = csign(x2-x1);
s2 = csign(y2-y1);
xchange = 0;
if (dy>dx)
{
    temp = dx;
    dx = dy;
    dy = temp;
    xchange = 1;
}
e = ((int)dy<<1) - dx;
for (j=0; j<=dx; j++)
{
    video_pt(x,y,c) ;
    if (e>=0)
    {
        if (xchange==1) x = x + s1;
        else y = y + s2;
        e = e - ((int)dx<<1);
    }
    if (xchange==1) y = y + s2;
    else x = x + s1;
    e = e + ((int)dy<<1);
}
}

//=====
//return the value of one point
//at x,y with color 1=white 0=black 2=invert
char video_set(char x, char y)
{
    //The following construction
    //detects exactly one bit at the x,y location
    i=((int)x>>3) + ((int)y<<3) ;
    return ( screen[i] & 1<<(7-(x & 0x7))) ;
}

//=====
void main(void)
{
    // initialize variables
    x = 30;
    y = 50;
    x1 = 30;
    y1 = 50;
    video_pt(x,y,1);
    pixFlag = 0;
    pixelSet = 0;
}

```



```

input = 0;

//setup uART for receiving
UCSRB = 0x10;          // set uART to receive
UBRR = 103;            // set baud rate to 4800

//init timer 1 to generate sync
OCR1A = lineTime;      //One NTSC line
TCCR1B = 9;            //full speed; clear-on-match
TCCR1A = 0x00;         //turn off pwm and oc lines
TIMSK = 0x10;         //enable interrupt T1 cmp

//init ports
DDRA = 0xf0;          //video out and switches
//D.5 is sync:1000 ohm + diode to 75 ohm resistor
//D.6 is video:330 ohm + diode to 75 ohm resistor
DDRC = 0x00;
PORTC = 0;

//init timer 0 to 1/uSec
TCCR0 = 2;

//initialize synch constants
LineCount = 1;
syncON = 0b00000000;
syncOFF = 0b00100000;

//side lines
video_line(0,0,0,99,1);
video_line(63,0,63,99,1);

//top line & bottom lines
video_line(0,0,63,0,1);
video_line(0,99,63,99,1);

//enable sleep mode
MCUCR = 0b01000000;
#asm ("sei");

//The following loop executes once/video line during lines
//1-230, then does all of the frame-end processing
while(1)
{
    //stall here until next line starts
    //sleep enable; mode=idle
    //use sleep to make entry into sync ISR uniform time

    #asm ("sleep");

    //Put code here to execute once/line
    //During the active portion of a line;
    //--Usable lines 1 to about 240

    if (LineCount<ScreenBot && LineCount>=ScreenTop)
    {
        //precompute pixel index for next line

```

```

//left-shift 3 would be individual lines
// <<2 means line-double the pixels
//The 0xffff8 truncates the odd line bit
i=(LineCount-ScreenTop)<<2 & 0xffff8; //0xffff8

//load the pixels into registers
//set up the computation, then use asm to speed it up

//v1 = screen[i];
//v2 = screen[++i];
//v3 = screen[++i];
//v4 = screen[++i];
//v5 = screen[++i];
//v6 = screen[++i];
//v7 = screen[++i];
//v8 = screen[++i];

#asm
push r26
push r27

ldi r26,low(_screen) ;base address of screen
ldi r27,high(_screen)
add r26,r12 ;offset into screen (add i)
adc r27,r13
ld r4,x ;load v1
adiw r26,1 ;inc offset to next screen byte
ld r5,x
adiw r26,1
ld r6,x
adiw r26,1
ld r7,x
adiw r26,1
ld r8,x
adiw r26,1
ld r9,x
adiw r26,1
ld r10,x
adiw r26,1
ld r11,x

pop r26
pop r27
#endasm

//now blast them out to the screen
PORTA.6=v1 & 0b10000000;
PORTA.6=v1 & 0b01000000;
PORTA.6=v1 & 0b00100000;
PORTA.6=v1 & 0b00010000;
PORTA.6=v1 & 0b00001000;
PORTA.6=v1 & 0b00000100;
PORTA.6=v1 & 0b00000010;
PORTA.6=v1 & 0b00000001;

PORTA.6=v2 & 0b10000000;
PORTA.6=v2 & 0b01000000;

```

PORTA.6=v2 & 0b00100000;
PORTA.6=v2 & 0b00010000;
PORTA.6=v2 & 0b00001000;
PORTA.6=v2 & 0b00000100;
PORTA.6=v2 & 0b00000010;
PORTA.6=v2 & 0b00000001;

PORTA.6=v3 & 0b10000000;
PORTA.6=v3 & 0b01000000;
PORTA.6=v3 & 0b00100000;
PORTA.6=v3 & 0b00010000;
PORTA.6=v3 & 0b00001000;
PORTA.6=v3 & 0b00000100;
PORTA.6=v3 & 0b00000010;
PORTA.6=v3 & 0b00000001;

PORTA.6=v4 & 0b10000000;
PORTA.6=v4 & 0b01000000;
PORTA.6=v4 & 0b00100000;
PORTA.6=v4 & 0b00010000;
PORTA.6=v4 & 0b00001000;
PORTA.6=v4 & 0b00000100;
PORTA.6=v4 & 0b00000010;
PORTA.6=v4 & 0b00000001;

PORTA.6=v5 & 0b10000000;
PORTA.6=v5 & 0b01000000;
PORTA.6=v5 & 0b00100000;
PORTA.6=v5 & 0b00010000;
PORTA.6=v5 & 0b00001000;
PORTA.6=v5 & 0b00000100;
PORTA.6=v5 & 0b00000010;
PORTA.6=v5 & 0b00000001;

PORTA.6=v6 & 0b10000000;
PORTA.6=v6 & 0b01000000;
PORTA.6=v6 & 0b00100000;
PORTA.6=v6 & 0b00010000;
PORTA.6=v6 & 0b00001000;
PORTA.6=v6 & 0b00000100;
PORTA.6=v6 & 0b00000010;
PORTA.6=v6 & 0b00000001;

PORTA.6=v7 & 0b10000000;
PORTA.6=v7 & 0b01000000;
PORTA.6=v7 & 0b00100000;
PORTA.6=v7 & 0b00010000;
PORTA.6=v7 & 0b00001000;
PORTA.6=v7 & 0b00000100;
PORTA.6=v7 & 0b00000010;
PORTA.6=v7 & 0b00000001;

PORTA.6=v8 & 0b10000000;
PORTA.6=v8 & 0b01000000;
PORTA.6=v8 & 0b00100000;
PORTA.6=v8 & 0b00010000;
PORTA.6=v8 & 0b00001000;

```

PORTA.6=v8 & 0b000000100;
PORTA.6=v8 & 0b000000010;
PORTA.6=v8 & 0b000000001;

PORTA.6=0 ;

}

//The following code executes during the vertical blanking
//Code here can be as long as 63 uSec/line
//For a total of 30 lines x 63.5 uSec/line x 8 cycles/uSec

if (LineCount==231)
{
    // receive 10101010, 3 times, to ensure proper gain of signal
    // aka receives 24 alternating 1's and 0's
    // ignore the first 24 bits received that were for the gain control
    input = UDR;

    // decode received instructions
    up = (input & 0x08) >> 3;    // up direction
    down = (input & 0x04) >> 2;  // down direction
    left = (input & 0x02) >> 1;  // left direction
    right = (input & 0x01);      // right direction
    Lbutt = (input & 0x20) >> 5; // draw
    Rbutt = (input & 0x10) >> 4; // erase
    Mbutt = (input & 0x40) >> 6; // clear screen

    // save previous x,y values
    x1 = x;
    y1 = y;

    // set new direction point to which mouse moved
    if (up && y < 98)          y++;
    else if (down && y > 1)     y--;
    if (left && x > 1)          x--;
    else if (right && x < 62)    x++;

    // draw or erase the points upon which the mouse is
    // draw point
    if (Lbutt)
    {
        video_pt(x,y,1);
    }
    // erase point
    else if(Rbutt)
    {
        video_pt(x,y,0);
    }
    // clear screen
    else if(Mbutt)
    {
        for (i = 0; i < 800; i++)
        {
            screen[i] = 0;
        }
    }

    //side lines
    video_line(0,0,0,99,1);
}

```

```

        video_line(63,0,63,99,1);

//top line & bottom lines
        video_line(0,0,63,0,1);
        video_line(0,99,63,99,1);

        // put the mouse cursor on the screen again
        x = 30;
        y = 50;
        video_pt(x,y,1);
    }
    // just move the mouse point
    else
    {

        if(pixFlag) {}
        else {video_pt(x1,y1,0);}

        pixelSet = video_set(x,y);

        if(pixelSet) {video_pt(x,y,1); pixFlag = 1;}
        else {video_pt(x,y,1); pixFlag = 0;}

    }

    } //line 231
} //while
} //main

```