A

Gig-A-Sketch

A Modified Etch-A-Sketch

EECS 129
Senior Design Project
Professor Raymond Klefstad

Robin Boaz
Thomas Bundick
Vinh Tran

Group I

Using an Etch-A-Sketch can be very frustrating; trying to create a drawing with two small knobs. Wouldn't it be easier to use a joystick to create your Etch-A-Sketch drawing. With the Gig-A-Sketch you can. It allows you to draw as easily as you ever had by steering your way through the drawing with an easy to use joystick.

2) Project Description

The Gig-A-Sketch is a modified Etch-A-Sketch that allows the user to control the drawing cursor with a wireless joystick. Instead of only being able to move in four directions (up, down, left or right) like the original Etch-A-Sketch, the user will be able to move in roughly 16 different directions. On the original, there are two knobs on the bottom that control the direction of the drawing cursor. One knob moves the cursors in strictly an up and down motion and the other knob moves the cursor in a strictly left and right motion. The Gig-A-Sketch will use the knobs to make the cursor move in any desired direction.

This design will use two microcontrollers to control the system. One microcontroller will interpret the movements of the joystick and send the interpreted movements wirelessly to another microcontroller that is attached to two motors controlling the knobs on the Etch-A-Sketch.

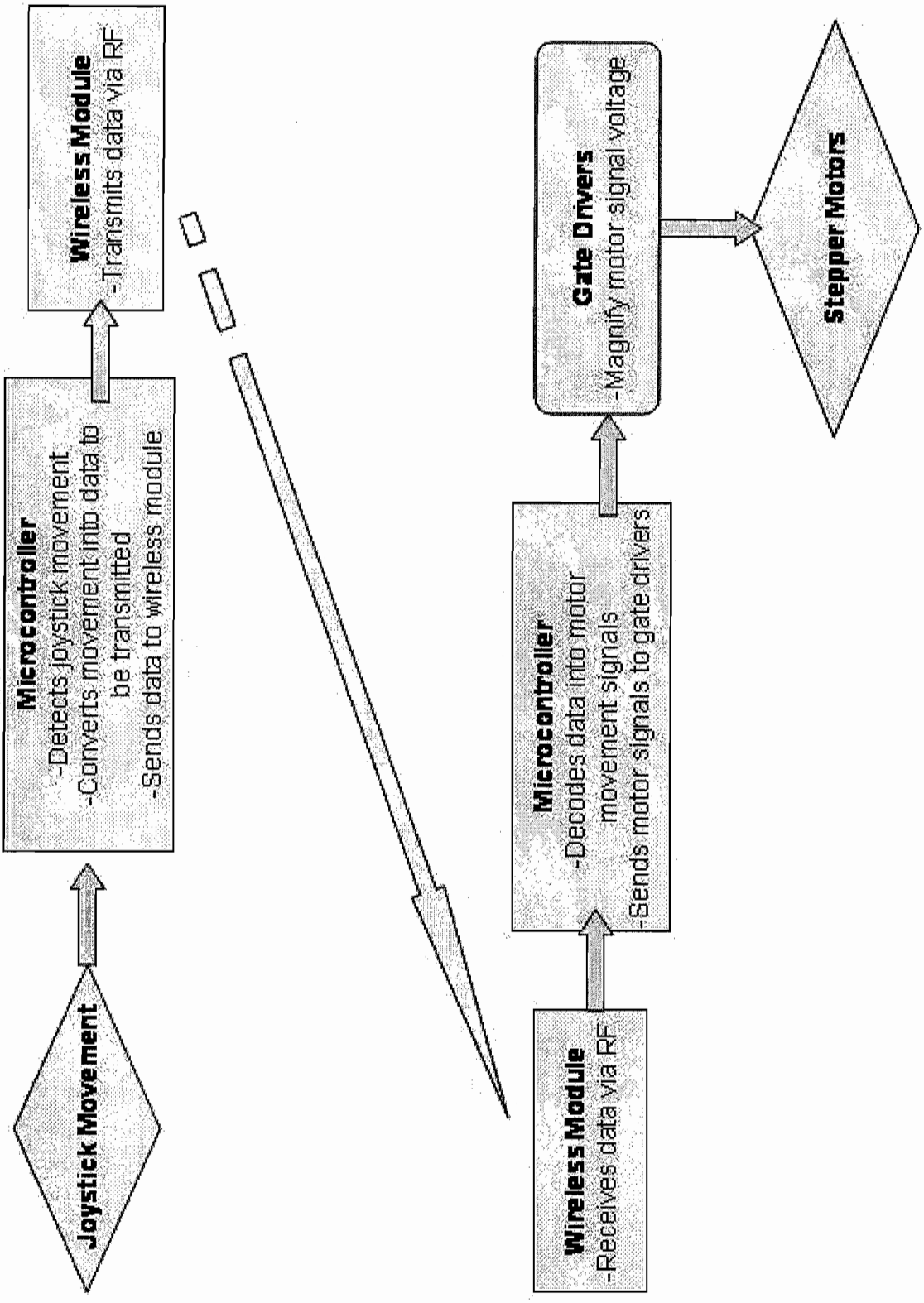
Instead of the knobs, the main drawing instrument will be the joystick. The joystick will move in a certain fashion, be interpreted by the microcontrollers, and then will move the knobs in order to draw the desired picture. The knob movements will be controlled by two stepper motors. The combined movements of the two motors will generate the directional movement of the cursor. Using simple geometry the 16 directions will be derived and set to move based on the direction of the joystick. Not only does the joystick control the direction, but it also controls the speed of the cursor. The speed depends on the magnitude of displacement from the equilibrium position of the joystick. In other words, the further away the joystick is from the center, the faster the cursor will move on the screen. Both the speed and directional calibration will be hard-coded into the Gig-A-Sketch.

The user will have a potentiometer joystick to which they can use to move in any direction, as they move the joystick, the resistive values of the X and Y potentiometers will vary. Also to note, the joystick is of analog nature, meaning that as the user presses the joystick further from the initial point the resistive values will increase or decrease linearly (depending on direction). Using these varying resistive nodes we can connect them to a known DC voltage source in series with a known resistor value and using our knowledge of voltage dividers find the varying value of voltage which will be input to the analog microcontroller pins. Using this method for an X and Y axis will allow the calculation of diagonals based on some simple geometric principle programming. The circuit is also connected to a ZigBee wireless transceiver which will send out the operations for the motor/receiver to receive and handle (explained in detail later).

Once the user has moved the joystick and created a signal as described above a signal is sent out and received on etch-a-sketch portion of the schematic. The signal will be received by another zigbee receiver which will then be sent through an IC buffer and finally to a digital pin of the microcontroller. The microcontroller will also have two stepper motors connected, two digital pins to each motor, this will allow for voltage to be sent in two directions causing the motor to be able to rotate both ways; and thus drive the etch-a-sketch handles. The direction and number of rotation the motors make will be dependent upon the programming of the microcontrollers.

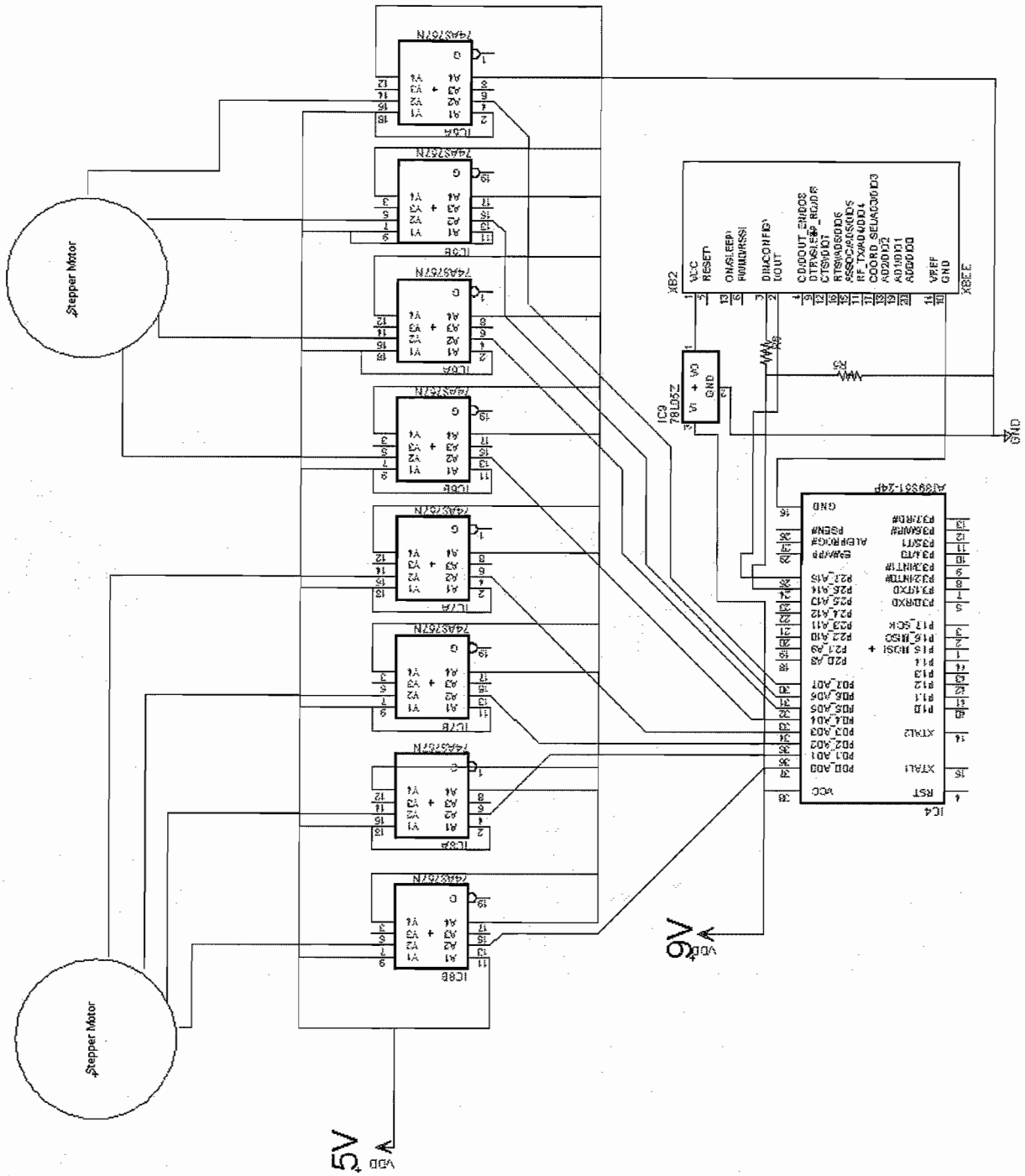
The Gig-A-Sketch is designed for use by people of all ages. It should be very easy to use and take almost no time to learn. With hardly any practice, someone should be able to pick it up rather quickly and draw simple pictures; and with much practice, anything should be able to be drawn using the Gig-A-Sketch.

3) System Level Block Diagram

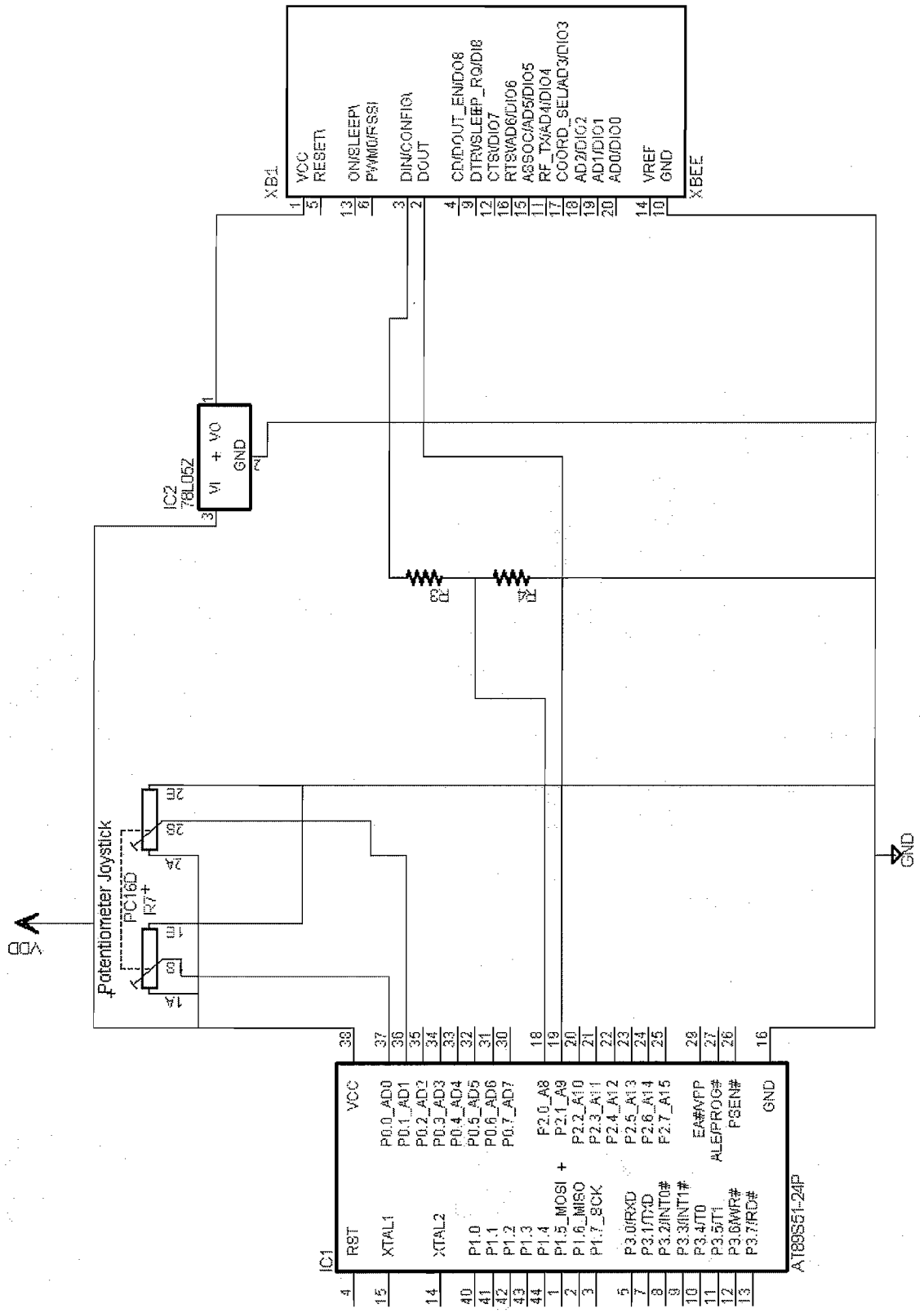


4) Circuit Level Block Diagram

Receiver Schematic:



Transmitter Schematic:



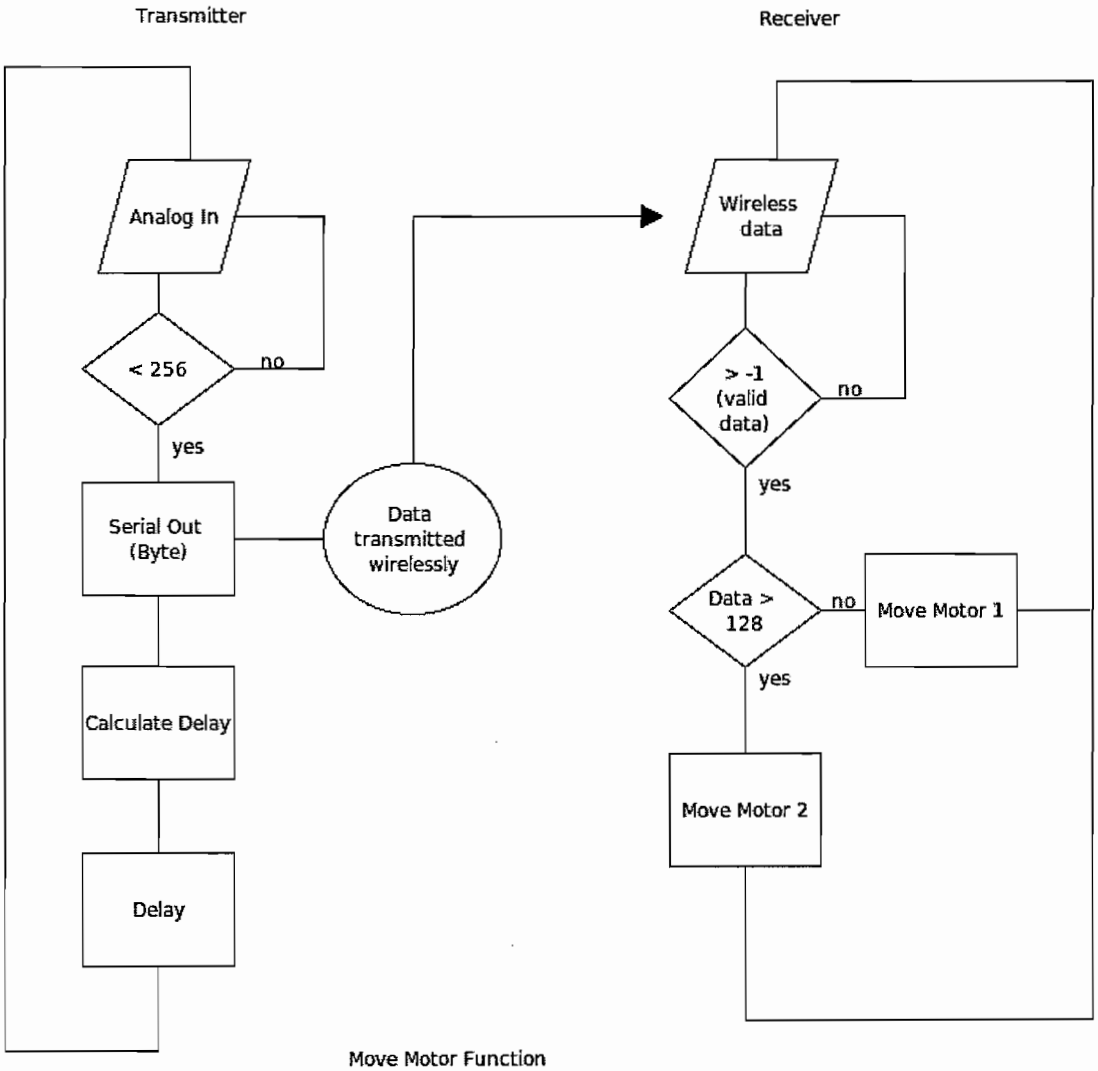
5) Software Description

Transmitter:
This works by first reading in an analog value. The analog value is generated by the voltage out of the joy stick. Once the analog value reaches a certain threshold it will then be ready to be transmitted. Before the value can be transmitted is must first be manipulated; the value is manipulated depending on which direction the joy stick is moved. This is done so the receiver knows which direction to move the motors. The new value will then be transmitted out, in byte form, using the serial print functionality provided by the Arduino microcontroller. Next a delay is calculated based on the new value. This is done so the transmitter and the receiver stay in sink, and the buffer's done have unwanted data. After the delay is finished the analog value is read in again, and the whole operation takes place again. Please not that no data is transmitted if the joy stick has not been moved.

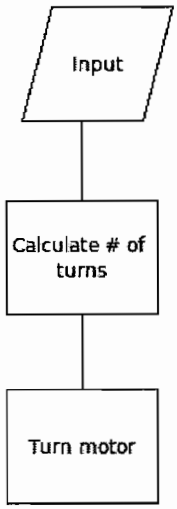
Receiver:
This works by doing a circular wait for a serial input. Once an input is received it is then processed. The processing of the value begins by seeing which motor group it fits in. If the value is between 1 to 128 it fits in the motor group one, and if the value is between 129 to 256 it will be in the motor group two. We can ensure that the value will land somewhere in that range because we are receiving byte data. There is no motor group in the code, and I am just using it so you can better understand what I am talking about; it is more of a bandwidth for each motor. Continuing, depending on which group the value fits in is which motor will be controlled. The motor is controlled by sending a the value modulated by 128 to a motor control function. This function controls the motors by taking the value in and calculating the number of turns the motor should do and determining the direction the motor should turn. Once the number of turns is calculated the motor is turned that many times in the desired direction by sending patterns of high and low output values to four different outputs in a specified pattern. After everything is complete the program goes back into its circular wait.

A flow chart is provided below to help you better understand the programs:

Gig-A-Sketch



Move Motor Function



6) System Test Plan

Stepper Motors' Reaction to Joystick

- No wireless
- Move joystick to control direction: up, down, left, right
- Control speed with varied force on the joystick
- Move joystick diagonally to alternate movement between both stepper motors
- Verify stepper motors control the Etch-a-Sketch knobs

Complete Functionality with No Wireless

- Verify everything works when both microcontrollers are connected to each other
- Test stepper motors' reactions to joystick movement
- Test reading/writing to memory

Wireless Functionality

- Send signal from the transmitter Zigbee unit and verify the receiving Zigbee unit receives the signal
- Test the stepper motors' reactions to the joystick by moving it and verifying that it controls the stepper motors to move in a certain direction

7) Cost Analysis

1 x Joystick Controller (potentiometer): POT JOYSTICK 10K OHM W/SWITCH
Manufacturer: CTS Corporation
Supplier: DigiKey Part#: 252A103B60NB-ND
$6.09

2 x ATmega8 USB board (microcontroller):
Manufacturer: Arduino
Supplier: SparkFun Part#: Arduino-USB
$31.95 each

2 x Stepper Motor: STEPPER MOTOR 5VDC 7.5DEG BI
Manufacturer: Portescap Danaher Motion US LLC
Supplier: DigiKey Part#: 403-1006-ND
$18.90 each

2 x Wireless Transmitter/Receiver: ZIGBEE 1MW W/WIRE ANT
Manufacturer: MaxStream Inc
Supplier: DigiKey Part#: XB24-AWI-001-ND
$19.00 each

2 x IC LDO REGULATOR +3.3V TO-220
Manufacturer: STMicroelectronics
Supplier: DigiKey Part#:497-1491-5-ND
$0.77 each

8 x Resistor: RES 10K OHM .50W 5% MF FUSIBLE
Manufacturer: Phoenix Passive Components
Supplier: DigiKey Part#: PPC10KBTR-ND
$0.07 each

8x IC MOSFET DRIVER LS 4A DUAL 8DIP
Manufacturer: IXYS
Supplier: DigiKey Part#: IXDN404PI-ND
$2.06 each

4x LED SS ALLNGAP ORN CLR 1.9MM GW
Manufacturer: Fairchild
Supplier DigiKey Part#: HLMAQH00AGR-ND
$0.17 each

1x Etch-A-Sketch
Manufacturer: Ohio Art
$14.95 each

Estimated Total Cost: **$180.00**

8) Summary

The Gig-A-Sketch works successfully. It successfully draws on the Etch-A-Sketch when we move the joystick. The signal generated by the transmitter module transmits successfully over the Zigbee module; the receiver unit receives this signal and interprets it into movements onto the appropriate motors.

One of the challenges that we encountered was that the gate drivers connecting to the motors were heating up and eventually burned out. These drivers could not take that much current needed to drive the motors. We resolved this by replacing them with new, more powerful drivers that can take more current. There was also an error in the software coding that caused this to happen. In the end, the problem has been resolved.

Another challenged we came across was that once the cursor is past a certain point on the Etch-A-Sketch, the cursor fails to draw. The motors move the knobs, but they do not generate enough torque to move the cursor. There is too much tension on the Etch-A-Sketch for the motors to move. We concluded that this problem is purely mechanical and that it out of our realm to resolve.

Our stepper motors are quite limited. When we draw diagonal lines, it turns out to be small staircases instead of straight diagonal lines. The stepper motors cannot get anymore precise. If we get better stepper motors, diagonal lines could be drawn straight on the Etch-A-Sketch.

All in all, the Etch-A-Sketch works to the best of our ability. With a little bit of practice, we can draw anything on this!

Gig-A-Sketch

Appendix

Transmitting code:

```
// an output light is on port 11
int outputPin = 11;

// a status light is on port 13
int led1 = 8;
int led2 = 9;

int input1, input2;

// a byte to receive data:
char inByte = 0;

int speakerOut = 11;
int button = 7;
//int speakerOut = 9;
//int button = 10;

byte names[] ={'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C'};
int tones[] = {1915, 1700, 1519, 1432, 1275, 1136, 1014, 956};
byte val = 0;
int serByte = -1;
int statePin = LOW;
int count = 0;
int buttonPushed = 0;

void setup () {
  // set pins to input and output appropriately
  pinMode(speakerOut, OUTPUT);
  pinMode(button, INPUT);
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);

  digitalWrite(led1,HIGH);
  delay(100);
  digitalWrite(led1, LOW);
  delay(100);

  // start up the serial connection with 9600-8-n-1-true (non-
inverted):
  Serial.begin(9600);


  Serial.print("X");
  delay(1100);

  digitalWrite(led2,HIGH);
  delay(100);
  digitalWrite(led2, LOW);
  delay(100);
  digitalWrite(led1,HIGH);
  delay(100);
  digitalWrite(led1, LOW);
```

```
  delay(100);

  // put the XBee in command mode
  Serial.print("+++");
  delay(1100);
  // wait for a response from the XBee for 2000 ms, or start
  // over with setup if no valid response comes

  if (returnedOK() == 'T') {
    // if an OK was received then continue
  }
  else {
    setup(); // otherwise go back and try setup again
  }

  digitalWrite(led1,HIGH);
  delay(100);

  // set the PAN (personal area network) ID number
  // this example uses 0x3330, but you'll want to choose your own
  // unique hexadecimal number between 0x0 and 0xFFFE
  // (note the comma at the end of the command which indicates that
another command will follow)
  Serial.print("ATID1234,CN\n");//DH0,DL1234,

  if (returnedOK() == 'T') {
    // if an OK was received then continue
  }
  else {
    setup(); // otherwise go back and try setup again
  }

}


void loop () {
  input1 = analogRead( 0 );
  input2 = analogRead( 1 );

  Serial.print( 'X' );
  Serial.print( '\n' );
  Serial.print( input1 );
  Serial.print( '\n' );
  delay(100);
  Serial.print( 'Y' );
  Serial.print( '\n' );
  Serial.print( input2 );
  Serial.print( '\n' );
  delay(100);
}


char returnedOK () {
  // this function checks the response on the serial port to see if it
was an "OK" or not
  char incomingChar[3];
  char okString[] = "OK";
```

```
  char result = 'n';
  int startTime = millis();
  while (millis() - startTime < 2000 && result == 'n') {  // use a
timeout of 10 seconds
    if (Serial.available() > 1) {
      // read three incoming bytes which should be "O", "K", and a
linefeed:
      for (int i=0; i<3; i++) {
        incomingChar[i] = Serial.read();
      }
      if ( strstr(incomingChar, okString) != NULL ) { // check to see
if the respose is "OK"
//      if (incomingChar[0] == 'O' && incomingChar[1] == 'K') { //
check to see if the first two characters are "OK"
        result = 'T'; // return T if "OK" was the response
      }
      else {
        result = 'F'; // otherwise return F
      }
    }
  }
  return result;
}
```

Receiver Code:

```
int pinRed1 = 7;
int pinGray1 = 6;
int pinYel1 = 5;
int pinBlk1 = 4;

int pinRed2 = 11;
int pinGray2 = 10;
int pinYel2 = 9;
int pinBlk2 = 8;

int turn = 4;

int dTime = 15;
int pattern[4][4] = { { HIGH, LOW, HIGH, LOW },
                      { HIGH, LOW, LOW, HIGH },
                      { LOW, HIGH, LOW, HIGH },
                      { LOW, HIGH, HIGH, LOW }};

int posM1 = 0;
int posM2 = 0;

int input1 = 0;
int input2 = 0;

// a byte to receive data:
char inByte = 0;

byte names[] ={'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C'};
int tones[] = {1915, 1700, 1519, 1432, 1275, 1136, 1014, 956};
byte val = 0;
```

```
int serByte = -1;
int statePin = LOW;
int count = 0;
int buttonPushed = 0;

void controlMotor1 ( int input ) {
  int i;

  /* when we have enough power use this
  digitalWrite(pinRed1,  pattern[posM1][0]);
  digitalWrite(pinGray1, pattern[posM1][1]);
  digitalWrite(pinYel1,  pattern[posM1][2]);
  digitalWrite(pinBlk1,  pattern[posM1][3]);


  digitalWrite(pinRed1,  LOW);
  digitalWrite(pinGray1, LOW);
  digitalWrite(pinYel1,  LOW);
  digitalWrite(pinBlk1,  LOW);
  */

  if ( input < 55 ) {  // clock wise
    for( i = 0; i < turn; i++ ) {
      digitalWrite(pinRed1,  pattern[posM1][0]);
      digitalWrite(pinGray1, pattern[posM1][1]);
      digitalWrite(pinYel1,  pattern[posM1][2]);
      digitalWrite(pinBlk1,  pattern[posM1][3]);
      delay(dTime);

      posM1 = (posM1 + 1 ) % 4;
    }
  }
  else if ( input > 75 ) {
    for( i = 0; i < turn; i++ ) {
      digitalWrite(pinRed1,  pattern[posM1][0]);
      digitalWrite(pinGray1, pattern[posM1][1]);
      digitalWrite(pinYel1,  pattern[posM1][2]);
      digitalWrite(pinBlk1,  pattern[posM1][3]);
      delay(dTime);

      posM1 = (posM1 - 1 );

      if ( posM1 > 3 )
        posM1 = 0;
      else if ( posM1 < 0 )
        posM1 = 3;
    }
  }

  /*
  switch ( abs((input-64) / 16) ) {
    case 0:
      delay( 100 );
    break;

    case 1:
      delay( 75 );
```

```
      break;

      case 2:
        delay( 50 );
      break;

      case 3:
        delay( 25 );
      break;
    }
    */

    digitalWrite(pinRed1,  LOW);
    digitalWrite(pinGray1, LOW);
    digitalWrite(pinYel1,  LOW);
    digitalWrite(pinBlk1,  LOW);


    Serial.print( '1' );
    Serial.print( '\t' );
    Serial.print( abs((input-64) / 16 ));
    Serial.print( '\n' );

}

void controlMotor2 ( int input ) {
  int i;

  /* when we have enough power
  digitalWrite(pinRed2,  pattern[posM2][0]);
  digitalWrite(pinGray2, pattern[posM2][1]);
  digitalWrite(pinYel2,  pattern[posM2][2]);
  digitalWrite(pinBlk2,  pattern[posM2][3]);


  digitalWrite(pinRed2,  LOW);
  digitalWrite(pinGray2, LOW);
  digitalWrite(pinYel2,  LOW);
  digitalWrite(pinBlk2,  LOW);
  */

  if ( input < 50 ) {  // clock wise
    for( i = 0; i < turn; i++ ) {
      digitalWrite(pinRed2,  pattern[posM2][0]);
      digitalWrite(pinGray2, pattern[posM2][1]);
      digitalWrite(pinYel2,  pattern[posM2][2]);
      digitalWrite(pinBlk2,  pattern[posM2][3]);
      delay(dTime);

      posM2 = (posM2 + 1 ) % 4;
    }
  }
  else if ( input > 70 ) {
    for( i = 0; i < turn; i++ ) {
      digitalWrite(pinRed2,  pattern[posM2][0]);
      digitalWrite(pinGray2, pattern[posM2][1]);
      digitalWrite(pinYel2,  pattern[posM2][2]);
```

```
        digitalWrite(pinBlk2,  pattern[posM2][3]);
        delay(dTime);

        posM2 = (posM2 - 1 );

        if ( posM2 > 3 )
          posM2 = 0;
        else if ( posM2 < 0 )
          posM2 = 3;
      }
    }

    /*
    switch ( abs((input-64) / 16) ) {
      case 0:
        delay( 100 );
      break;

      case 1:
        delay( 75 );
      break;

      case 2:
        delay( 50 );
      break;

      case 3:
        delay( 25 );
      break;
    }
    */

    digitalWrite(pinRed2,  LOW);
    digitalWrite(pinGray2, LOW);
    digitalWrite(pinYel2,  LOW);
    digitalWrite(pinBlk2,  LOW);


    Serial.print( '2' );
    Serial.print( '\t' );
    Serial.print( abs((input-64) / 16 ));
    Serial.print( '\n' );

}

void setup () {
  pinMode(pinRed1, OUTPUT);
  pinMode(pinGray1, OUTPUT);
  pinMode(pinYel1, OUTPUT);
  pinMode(pinBlk1, OUTPUT);

  pinMode(pinRed2, OUTPUT);
  pinMode(pinGray2, OUTPUT);
  pinMode(pinYel2, OUTPUT);
  pinMode(pinBlk2, OUTPUT);

  // start up the serial connection with 9600-8-n-1-true (non-
```

```
inverted):
  Serial.begin(9600);


  Serial.print("X");
  delay(1100);

  // put the XBee in command mode
  Serial.print("+++");
  delay(1100);
  // wait for a response from the XBee for 2000 ms, or start
  // over with setup if no valid response comes

  if (returnedOK() == 'T') {
    // if an OK was received then continue
  }
  else {
    setup(); // otherwise go back and try setup again
  }

  // set the PAN (personal area network) ID number
  // this example uses 0x3330, but you'll want to choose your own
  // unique hexadecimal number between 0x0 and 0xFFFE
  // (note the comma at the end of the command which indicates that
another command will follow)
  Serial.print("ATID4321,CN\n");//DH0,DL1234,

  if (returnedOK() == 'T') {
    // if an OK was received then continue
  }
  else {
    setup(); // otherwise go back and try setup again
  }

}

int inputValue = 0;
int count1 = 0;

void loop () {
  if (Serial.available() > 1) {
    serByte = Serial.read();
    if (serByte != -1) {
      val = serByte;
      statePin = !statePin;
    }
    else {
      val = 'p';
    }

    /*
    if ( serByte != 13 ) {
      serByte = serByte << count1;
      count1++;

      inputValue += serByte;
    }
```

```
    else {
      Serial.print( inputValue );
      Serial.print( '\n' );

      count1 = 0;
      inputValue = 0;
    }


    Serial.print( serByte );
    Serial.print( '\n' );
    val = 'p';
    */

    if ( serByte > 128 ) {
      controlMotor2( serByte - 128 );
    }
    else
    if ( serByte ) {
      controlMotor1( serByte );
    }
  }

}

//1: 11 - 130 - 246
//2: 0 - 118 - 255

char returnedOK () {
  // this function checks the response on the serial port to see if it
was an "OK" or not
  char incomingChar[3];
  char okString[] = "OK";
  char result = 'n';
  int startTime = millis();
  while (millis() - startTime < 2000 && result == 'n') {  // use a
timeout of 10 seconds
    if (Serial.available() > 1) {
      // read three incoming bytes which should be "O", "K", and a
linefeed:
      for (int i=0; i<3; i++) {
        incomingChar[i] = Serial.read();
      }
      if ( strstr(incomingChar, okString) != NULL ) { // check to see
if the respose is "OK"
//      if (incomingChar[0] == 'O' && incomingChar[1] == 'K') { //
check to see if the first two characters are "OK"
        result = 'T'; // return T if "OK" was the response
      }
      else {
        result = 'F'; // otherwise return F
      }
    }
  }
  return result;
}
```