Wireless Etch-a-Sketch

Final Project Report
December 11, 2009
E155

Daniel Bujalski and Jaakko Karras

Abstract

The Etch-a-Sketch is a classic artistic toy that has an amazing capacity for expression. However, for a good many people, trying to produce quality images or even plain text on the device is a very difficult challenge. The only way to produce reliable results with the toy, short of having a very steady hand, is to use mechanical and electronic control to turn the knobs. Thus, our project's goal was to implement a system that uses such a control scheme to draw regular and recognizable alphanumeric characters on the Etch-a-Sketch by simply inputting the characters on a computer. A file containing a message is read and transmitted to the device, which then analyzes and draws each of the characters on the Etch-a-Sketch screen.

Introduction

The Etch-a-Sketch, first produced decades ago, has been a popular drawing tool for young and old alike. However, as many who have used it should know, it is not very easy to draw with. Even with only straight lines, trying to keep a drawing aligned is a tricky task, and anything curved or angled takes a lot of practice. Possibly as a result of this, there have been several cases of people controlling the Etch-a-Sketch through electronic means, oftentimes autonomously. This way, other forms of input can be used to draw with the toy.

Our project is of a similar form, using motors controlled by electronics to draw something that might be considered difficult on an Etch-a-Sketch. In order to greatly simplify the input, we decided to have the device draw from the contents of a single text file. This way, a user could write legible messages on the Etch-a-Sketch by typing them into a computer. In order to accomplish this, we pass the information from the input file through a wireless link to our electronics, which then decode and draw the resulting characters. The block diagram for the overall device is shown in *Figure 1* below. Specific components will be described in greater detail later on.

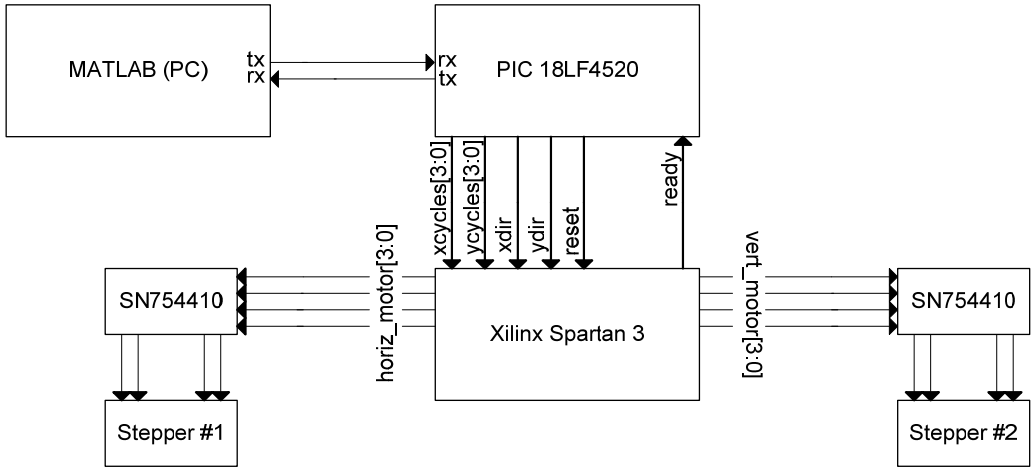

Figure 1. High-level block diagram for wireless Etch-a-Sketch system.

Using MATLAB, the PC opens a serial link over a Bluetooth connection to the PIC microcontroller on the Harris Board. Using this link, the PC sends each input character individually which the PIC then receives and processes before accepting another. Using a preloaded database of drawn points corresponding to each character, the PIC calculates where the Etch-a-Sketch needs to draw to next and transmits the results to the FPGA. Upon receiving

one of these "commands" from the PIC, the FPGA sends signals to a pair of H-bridges that drive the attached stepper motors accordingly.

In the following sections, the major components of our design as seen in the previous block diagram will be described in more detail. In addition, we have included schematics of onboard electronics, the results of our project, and the code sets for the FPGA, PIC, and MATLAB components.

New Components

This project incorporates a number of components that were not covered by the standard E155 laboratory assignments. These new components include MATLAB's serial I/O library, used for wireless communication between the PC and the PIC, and stepper motors, used in actuating the Etch-a-Sketch knobs. These components are presented in detail for those wishing to utilize these components in future projects.

MATLAB's Serial I/O Library

MATLAB's serial I/O library includes a number of built-in functions that facilitate serial communication [1]. Overall, the serial I/O library functions very much like file I/O; serial ports are opened, read from and written to in much the same way as files. The following functions provided the basic serial communications used in completing this project:

1. `serial( )` - This is the constructor for serial port objects. The constructor accepts a number of inputs, perhaps the most important of which are the COM port for the desired connection and the baud rate.

   Example: `ser = serial('COM86', 'BaudRate', 9600);`

   Once created, the serial port object `ser` is used to read from and write to the serial port. Additionally, serial port objects have a useful attribute, `BytesAvailable`, which returns the number of bytes accessible in the serial port's receive buffer.
2. `fopen( )` - This function opens a serial port object.

   Example: `fopen(ser);`
3. `fread( )` - This function reads up to a specified number of bytes from the serial port receive buffer.

Example: `Data = fread(ser, ser.BytesAvailable);`

4. `fwrite( )` - This function writes its input data to the specified serial port.

   Example: `fwrite(ser, Data);`

5. `fclose( )` - This function closes the specified serial port connection.

   Example: `fclose(ser);`

Stepper Motors and H-Bridge Circuitry

Stepper motors are electric motors that divide angular displacements into sequences of discrete steps [2]. Because of their stepping behavior, stepper motors provide an inexpensive and relatively straightforward solution for situations requiring control over angular position, as is the case with the Etch-a-Sketch system.

Beneath the casing, stepper motors contain some number of electrically actuated "phases". When a phase is activated by an externally applied current, it causes the rotor to align with it. This alignment produces a small rotation or "step" at the motor shaft. Hence, a stepper motor is driven by activating and deactivating the various phases in a sequential manner. The order in which the phases are activated and deactivated dictates the motor's direction of rotation.

Stepper motor phases are typically either uni- or bipolar. A unipolar phase is one with two windings, whereas a bipolar phase has a single winding. Activating and deactivating a unipolar phase involves applying a voltage to one of the two windings at a time. Driving a bipolar phase, on the other hand, requires an H-bridge circuit capable of reversing the direction of current through the single winding.

The stepper motors used for this project were 7.5º/step bipolar steppers. As such, they were driven using a pair of SN754410 H-bridge ICs. The wiring details for the H-bridge chips are provided in the schematics in the following section. It was observed that, although the SN754410 chips are rated for up to 1A, they heat up quickly at such high currents and ultimately become unstable. The solution employed for this project was to solder two H-bridge ICs together in parallel, attach a snap-on IC heat sink and run entire circuit at 4.5 volts, as opposed to the motors' 5 volt rating.

Schematics

The following two schematics detail the breadboard circuitry driving the Etch-a-Sketch.

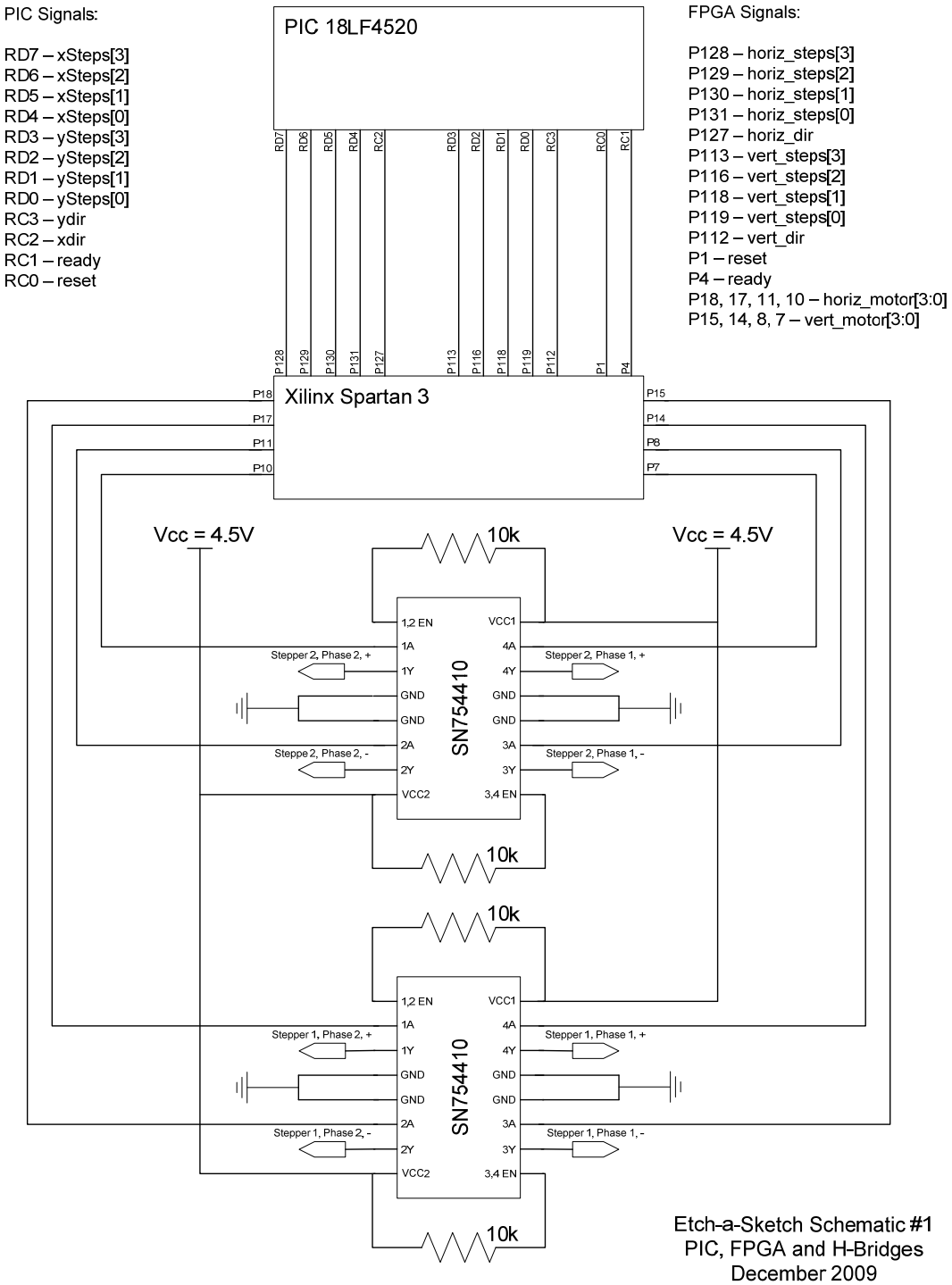

Figure 2. Schematic showing PIC, FPGA and H-bridge interaction.

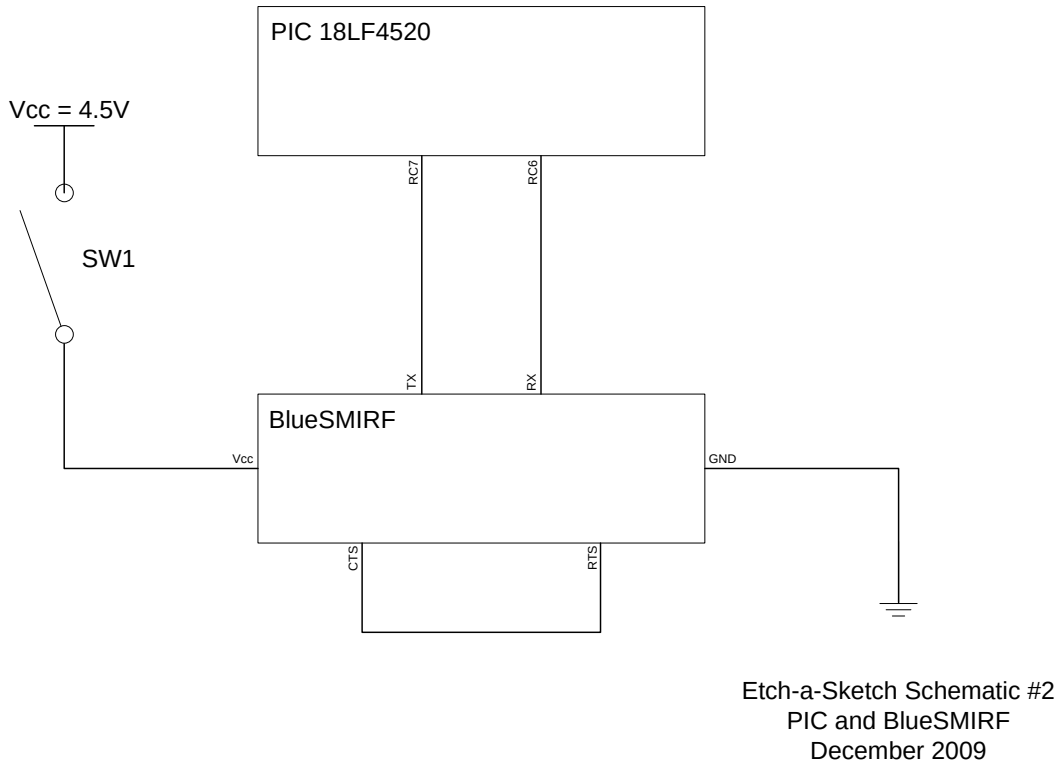

Figure 3. Schematic showing PIC and BlueSMIRF interaction.

PC Component

The PC is responsible for initiating the drawing process. When started, the MATLAB script (*Appendix A*) opens the text file containing the message and establishes a wireless connection with the PIC via the BlueSMIRF transceiver. Having established the wireless connection, MATLAB waits for the PIC to send it an 'R' character to indicate that it is ready to receive a new character.

Whenever MATLAB receives a ready signal from the PIC, it simply reads the next character from its input text file and transmits this to the PIC. When all the characters have been exhausted, the script closes its serial connections and halts.

PIC Microcontroller Component

The PIC18F4520 microcontroller on the Harris Board 2.0 serves as a translator for the device, receiving PC inputs and producing outputs that the FPGA uses to drive the Etch-a-Sketch motors.

To begin, on a reset the PIC continually sends the letter "R" through the Bluetooth serial link. This signifies that the PIC is ready for the next input character. Once it receives an input, the PIC stops transmitting the ready signal until the character has been processed and drawn.

In order to translate the alphanumeric character into Etch-a-Sketch drawing, the PIC uses a `switch` statement to choose one of many character draw functions - each one corresponding to different letter or number. Each character draw function contains several calls of the function `drawpt` that correspond to the relative points that must be connected to create the letter on the Etch-a-Sketch screen.

In order to understand `drawpt`, we need to know how the PIC handles screen positioning. Since the PIC has no direct feedback from the physical device, all drawing must be done relative to some point of origin. The program handles dimensions with an arbitrary pixels-to-steps scaling factor of five; that is, the PIC considers each step to be equal to five “pixels”. The characters to be drawn are 60 x 100 "pixels" in size, and are drawn in four rows of ten with spacing of 20 "pixels" between the letters and the rows. The PIC stores the current character position (row and column) as a pair of global variables `xcurr` and `ycurr`.

The function `drawpt` is key to converting this virtual positioning into FPGA commands, since it converts "pixel" motion into a number of steps for the horizonal and vertical motors to turn. The `drawpt` inputs are all in a form `drawpt(xcurr+x,ycurr+y)`, where `x` and `y` are the relative positions (from the character's origin `xcurr,ycurr`) of the next point to draw towards on screen.

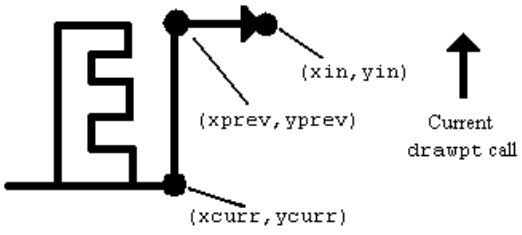

Figure 4. Example of `drawpt` in action with reference points labeled. (`xin`, `yin`) represents a point relative to (`xcurr`, `ycurr`).

The `drawpt` function compares its `(xin,yin)` input to the cursor's current position (stored using global variables `xprev` and `yprev`) to determine the horizontal and vertical movement required to reach the new point (as well as the directions in which to move). Then, the number of "pixels" to move for both dimensions is scaled down by five to convert the movements to steps. Some compensation is introduced here in the form of additional steps for a change of direction to account for Etch-a-Sketch slack.

At this point, the PIC waits for the FPGA to be ready (ie: finished drawing the previous command) by waiting for a designated "ready" input from the FPGA to be set. When this happens, the PIC passes the horizontal and vertical steps and directions to the FPGA through output ports and sets a "reset" output to make the FPGA begin drawing. After storing the current virtual position into `xprev` and `yprev`, the PIC returns to the current character draw function and moves to the next `drawpt` call. If these are exhausted, the character has been completed and the PIC changes the value of `xcurr` and `ycurr` for the next letter.

FPGA Component

The purpose of the Xilinx Spartan 3 FPGA in our project is to function as a low-level motor controller. It receives the motion inputs from the PIC for horizontal and vertical directions and outputs signals to the H-bridges that change the stepper motors' phases, thus drawing the picture.

On startup, the FPGA waits for the PIC to send a command for the next point to draw to. While it waits, it sets the previously-mentioned "ready" pin in order to inform the microcontroller that nothing is currently being drawn. The FPGA only actually receives the PIC's output if the microcontroller sets the "reset" pin which prevents reading the wrong values. After this occurs, two instances of the module `step_counter` place the cycle inputs into the `target` wire for each (one module for horizontal and one for vertical). As long as a counter variable is not equal to `target`, the counter will increment on a clock edge and hold an enable, `en`, high. As long as `en` is high, a pair of instances of `motor_fsm` will change the FPGA outputs to the two H-bridges connected to the two motors. The finite state machine in `motor_fsm` has four states that correspond to the four phases or positions that the motors

possess, and rotating through these phases in order through the state machine drives the motors in the proper directions. The state machine is shown in *Figure 5*.

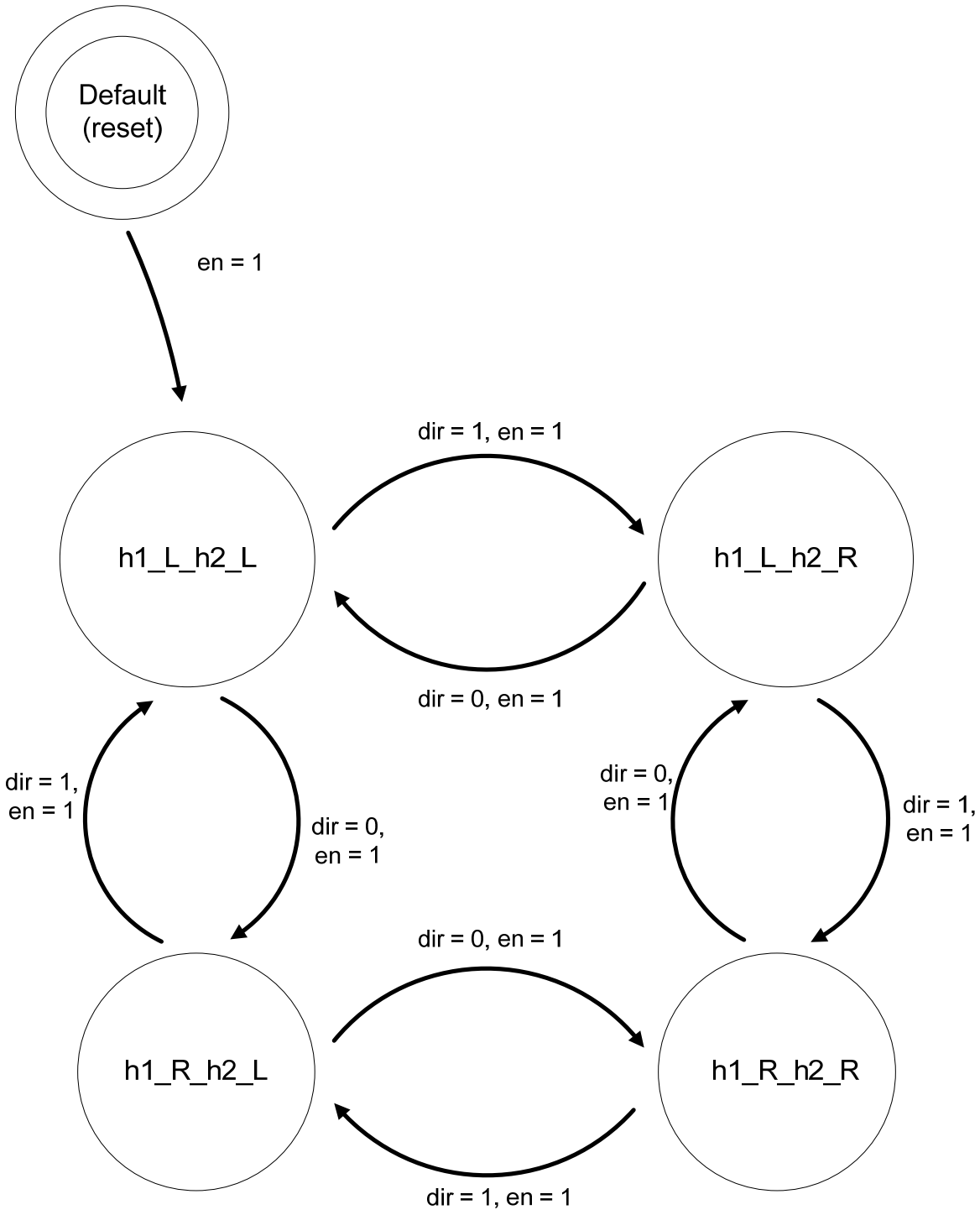

Figure 5. Finite state machine diagram for motor_fsm showing the four phases for each motor.

The result of this process is that the state machines for horizontal and vertical motion change state a number of times equal to the input number of steps for horizontal and vertical motion. This drives each motor the same number of phases forward or backward, depending on the input directions, which draws on the Etch-a-Sketch the desired line segment. Once this is complete and both `en` values are low, the FPGA sends its "ready" signal again.

Mechanical Component

The quality of the Etch-a-Sketch results is heavily influenced by the mechanical components used to mount the stepper motors to the toy. It was determined early on that any slippage in the stepper motor connections was detrimental to the accuracy of its sketches. The initial mechanical design called for attaching the stepper motor shafts to the Etch-a-Sketch knobs using shaft couplers and then attaching both the motors and the toy to a common piece of plywood using long machine screws. This design, however, was problematic for the simple reason that the Etch-a-Sketch knobs are not perfectly perpendicular to the casing and require a certain amount of “wobble” as they turn. In other words, it became important to restrict the stepper motors’ rotation along their own axes while leaving all other forms of motion as unrestricted as possible. If the steppers were not allowed to wobble, they would get stuck and skip drawing steps.

The final mechanical design involved mounting the steppers to the knobs as before, but instead of mounting the motors and Etch-a-Sketch to a common piece of wood, the steppers were attached to square corner brackets that hug the edges of the toy. These brackets are effective in preventing the motors from rotating relative to the toy, but allow sufficient vertical and lateral wobbling so that the motors do not become stuck.

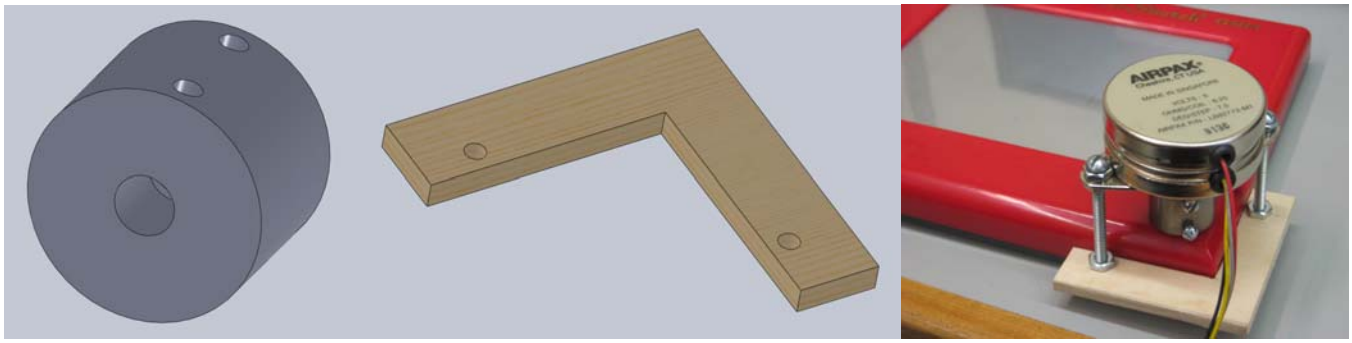

Figure 6. CAD representation of motor shaft coupler and restricting bracket and final product.

Results

For all intensive purposes, our device functioned exactly as desired. It drew any and all of the alphanumeric messages we provided in the input text file, and the characters were recognizable and legible.

There are, however, some concerns and slight issues that still exist with the project, although none of them seriously impair operation. First and foremost, the letters and numbers drawn were not perfect and still exhibited some drawing inaccuracy, particularly in the horizontal dimension. However, the effect did not affect readability or positioning and simply gave the characters a sort of "artistic style". Secondly, the MATLAB serial connection was a little finicky to get started in many cases. This too proved to be of little concern, as it usually took only one repeated attempt at most to establish the link. Finally, we did not implement an erasure mechanism or reset positioning for the stylus, meaning that after each run we were forced to reset the device by hand. This required disconnecting the motors in order to turn the knobs by hand. For convenience's sake, we would have liked to remedy this, but neither problem affected the device's output and thus, for the sake of time, they were not addressed.

Figure 7. Photograph of text drawn by Etch-a-Sketch.

References

1. "Getting Started with Serial I/O." The MathWorks. <http://www.mathworks.com/access/helpdesk/help/techdoc/matlab_external/f92576.html>.

2. "Stepper motor." Wikipedia. < http://en.wikipedia.org/wiki/Stepper_motor >.

3. SN754410 Quadruple Half-H Driver Data Sheet. Texas Instruments. November 1995. <http://www3.hmc.edu/~harris/class/e155/sn754410.pdf>.

Parts List

Mechanical Components

Item	**Qty**	**Supplier**	**Cost**
Classic Etch-a-Sketch	1	Amazon	$19.08
¼" Birch Plywood	1	Lowe's	$11.85
1" Cylindrical Steel Stock	1	Student Shop	$1.00
7.5º Stepper Motor	2	E155 Lab	$0.00
2 ½" 10-32 Machine Screws	4	Engr. Wall of Parts	$0.00
10-32 Nuts	16	Engr. Wall of Parts	$0.00
½" 6-32 Machine Screws	4	Engr. Wall of Parts	$0.00

Electrical Components

Item	**Qty**	**Supplier**	**Cost**
Harris Board	1	E155 Supplies	$0.00
BlueSMIRF Transceiver	1	E155 Lab	$0.00
USB Bluetooth Dongle	1	E155 Lab	$0.00
SN754410 H-Bridge	4	E155 Lab	$0.00
IC Snap-On Heat Sink	2	Engr. Stockroom	$0.00
10k Ohm Resistor	4	Personal Supplies	$0.00
SPDT Switch	1	E155 Lab	$0.00
LED	5	E155 Lab	$0.00
		Total:	**$31.93**

Appendix A: MATLAB Script

```
% Wireless Etch-a-Sketch Communication
%
% Reads in text file containing message to be transferred
% and carries out appropriate communication with PIC
% microcontroller over Bluetooth link.
%
% Text file should be limited to 40 characters (A-Z, 0-9).
% Characters will be drawn in four rows of ten.
%
% jkarras, dbujalski@hmc.edu
% December 2009

% Open file containing message
%     File stored under: /MATLAB/E155_Final_Project
message_file = 'message_file.txt';
fid = fopen(message_file, 'r');



% Establish connection to Etch-a-Sketch
%     Data, Parity, Stop: "8N1" (MATLAB default)
%     Baud Rate: 9600
%     Outgoing COM: <Refer to Bluetooth Device>

ser =  serial('COM86', 'BaudRate', 9600);
fopen(ser);

% Clear receive buffer
if ser.BytesAvailable ~= 0
    fread(ser, ser.BytesAvailable)
end

% Read through contents of message_file, sending characters to PIC
eof = feof(fid);
while eof ~= 1
    character = fread(fid, 1);
    eof = feof(fid);

    % Wait for PIC ready signal
    while ser.BytesAvailable == 0
    end
    fread(ser, ser.BytesAvailable)

    % Write index character to PIC
    fwrite(ser, character);

end

% Close handles
fclose(fid);
fclose(ser);
delete(ser);
```

Appendix B: PIC Microcontroller Code

```
/*  Wireless Etch-a-Sketch PIC Code

      Authors: dbujalski, jkarras@hmc.edu
      Created: November 2009

      Establishes communication between MATLAB and the FPGA, allowing
      characters to be transmitted to the PIC and drawn onto the Etch-
      a-Sketch.

*/


#include <p18f4520.h>
#include <stdio.h>
#include <stdlib.h>

#define CHARWIDTH = 80;
#define CHARHEIGHT = 120;


/*
FUNCTION DECLARATIONS
*/
void configUSART(void);
void drawNextChar(char);
void drawpt(unsigned short, unsigned short);
void drawA(void);
void drawB(void);
void drawC(void);
void drawD(void);
void drawE(void);
void drawF(void);
void drawG(void);
void drawH(void);
void drawI(void);
void drawJ(void);
void drawK(void);
void drawL(void);
void drawM(void);
void drawN(void);
void drawO(void);
void drawP(void);
void drawQ(void);
void drawR(void);
void drawS(void);
void drawT(void);
void drawU(void);
void drawV(void);
void drawW(void);
void drawX(void);
void drawY(void);
void drawZ(void);
void draw1(void);
void draw2(void);
```

```
void draw3(void);
void draw4(void);
void draw5(void);
void draw6(void);
void draw7(void);
void draw8(void);
void draw9(void);
void draw0(void);
void drawSpace(void);


/*
GLOBAL VARIABLES
*/
int i;
unsigned short xcurr = 0;     // Xcoord of lower left-hand reference
unsigned short ycurr = 480;   // Ycoord of lower left-hand reference
unsigned char charCount = 0;  // Number of characters currently on screen
unsigned short xprev = 0;     // Xcoord of last drawn point
unsigned short yprev = 480;   // Ycoord of last drawn point
unsigned short xdirprev = 0;  // Horiz direction of previous line
unsigned short ydirprev = 0;  // Vert direction of previous line


/* main()
*/
void main(void) {

      char receivedChar;      // Character received over Bluetooth

      TRISD = 0;              // For LED/FPGA output
      PORTD = 0;
      TRISC = 0x01;           // Bit 0: FPGA ready signal (input)
                              // Bit 1: FPGA reset (output)
                              // Bit 2: xdir, Bit 3: ydir (both outputs)
      PORTCbits.RC1 = 0;      // Hold PIC reset low (no data to send yet)

      /* CALIBRATION
         Fix motor positioning for start point to ensure FPGA state
         matches actual motor state (accounts for play in motors)
      */
      drawpt(10,480);
      drawpt(0,480);
      drawpt(0,490);
      drawpt(0,480);

      // Setup the serial connection
      configUSART();

      // Repeatedly request characters from MATLAB
      while(1)
      {
            // Request next character from MATLAB
            printf("R");

            // Wait for transmitted character
            while (!PIR1bits.RCIF);
```

```
			// Read character off of the FIFO
			receivedChar = RCREG;

			// Draw the character
			drawNextChar(receivedChar);
		}

	}


	/*
		configUSART()

		Establish an asynchronous serial connection between the PIC and the
		host PC. The PIC USART pins (TX, RX) will connect to the USART_RX,
		USART_TX pins on the BlueSMIRF.

		Connection Parameters:
			Baud Rate:		115.2k
			Data Bits:		8
			Stop Bits:		1
			Parity:			None
			Flow Control:		None
	*/
	void configUSART(void)
	{
		/*
		Configure TXSTA Register
			bit 7: 0 (don't care)
			bit 6: 0 (8-bit transmission)
			bit 5: 1 (enable transmit)
			bit 4: 0 (asynchronous mode)
			bit 3: 0 (unimplemented)
			bit 2: 1 (high speed -- lower BRG error)
			bit 1: 0 (read only)
			bit 0: 0 (don't care)
		*/
		TXSTA = 0x24;

		/*
		Configure RCSTA Register:
			bit 7: 1 (enable serial port)
			bit 6: 0 (8-bit reception)
			bit 5: 0 (don't care)
			bit 4: 1 (enable receiver)
			bit 3: 0 (don't care)
			bit 2: 0 (read only)
			bit 1: 0 (read only)
			bit 0: 0 (read only)
		*/
		RCSTA = 0x90;
```

```
	/*
	Configure Baud Rate Generator:
		Assume: 20MHz Fosc
				115.2k Baud Rate
				BRGH = 1 (high setting)
		Timer Period, X = (20,000,000)/(115,200 * 16) - 1
					X = 9.85, or 10

		Actual Baud Rate = 113,636
		Baud Rate Error = -1.35% (acceptable)
	*/
	SPBRG = 0x0A;
}


/* drawNexChar()

   Draws the most recently transmitted character.
*/
void drawNextChar(char character)
{

	// Decipher character input, convert to line segments
	switch (character)
	{
		case ' ':
			drawSpace();
			break;
		case 'A':
			drawA();
			break;
		case 'B':
			drawB();
			break;
		case 'C':
			drawC();
			break;
		case 'D':
			drawD();
			break;
		case 'E':
			drawE();
			break;
		case 'F':
			drawF();
			break;
		case 'G':
			drawG();
			break;
		case 'H':
			drawH();
			break;
		case 'I':
			drawI();
			break;
		case 'J':
			drawJ();
```

```
      break;
case 'K':
      drawK();
      break;
case 'L':
      drawL();
      break;
case 'M':
      drawM();
      break;
case 'N':
      drawN();
      break;
case 'O':
      drawO();
      break;
case 'P':
      drawP();
      break;
case 'Q':
      drawQ();
      break;
case 'R':
      drawR();
      break;
case 'S':
      drawS();
      break;
case 'T':
      drawT();
      break;
case 'U':
      drawU();
      break;
case 'V':
      drawV();
      break;
case 'W':
      drawW();
      break;
case 'X':
      drawX();
      break;
case 'Y':
      drawY();
      break;
case 'Z':
      drawZ();
      break;
case '1':
      draw1();
      break;
case '2':
      draw2();
      break;
case '3':
      draw3();
```

```
                break;
        case '4':
                draw4();
                break;
        case '5':
                draw5();
                break;
        case '6':
                draw6();
                break;
        case '7':
                draw7();
                break;
        case '8':
                draw8();
                break;
        case '9':
                draw9();
                break;
        case '0':
                draw0();
                break;
        default:
                drawSpace();
                break;
    }

    // Increment character counter and adjust Xcoord reference
    ++charCount;
    xcurr += 80;

    // Account for new line (every ten characters) by adjusting
    // Xcoord, Ycoord references
    if (!(charCount%10))
    {
        // Return to left margin
        for (i = 0; i < 10; ++i)
        {
            drawpt(xcurr - 40, ycurr);
            drawpt(xcurr - 80, ycurr);
            xcurr -= 80;
        }
        // Return to left margin, move down 120
        drawpt(xcurr, ycurr - 40);
        drawpt(xcurr, ycurr - 80);
        drawpt(xcurr, ycurr - 120);
        ycurr -= 120;
    }
}
```

```
/* drawpt()

   Inputs:
   1. xin, yin indicating next point
   2. ready flag (PORTC[0])from FPGA indicating need for next point

   Outputs:
   1. xSteps (PORTD[7:4]), ySteps (PORTD[3:0]) indicating the number of
      steps to take
   2. FPGA reset (PORTC[1])
   3. xdir (PORTC[2]), ydir (PORTC[3])

   Number Convention:
   xdir = 1 : LEFT
   xdir = 0 : RIGHT
   ydir = 1 : DOWN
   ydir = 0 : UP
*/
void drawpt(unsigned short xin, unsigned short yin) {

      unsigned short xchange = 0;   // Variables to store difference between
      unsigned short ychange = 0;   //    next and previous points

      unsigned char xSteps = 0;     // Number of steps to take along horiz.
      unsigned char ySteps = 0;     // Number of steps to take along vert.
      unsigned char xStepsComp = 0; // Compensated # of horizontal steps
      unsigned char yStepsComp = 0; // Compensated # of vertical steps

      char xdir = 0;                // Variables to store direction bits
      char ydir = 0;

      //Find difference in x and y coordinates and set directions

      /* Horizontal Steps */
      if(xin == xprev){
            xchange = 0;
            xdir = xdirprev;
      }
      else {
            if (xprev > xin){
                  xchange = xprev - xin;
                  xdir = 1;
            }
            else {
                  xchange = xin - xprev;
                  xdir = 0;
            }
      }

      /* Vertical Steps */
      if(yin == yprev){
            ychange = 0;
            ydir = ydirprev;
      }
      else {
            if (yprev > yin){
                  ychange = yprev - yin;
```

```
            ydir = 1;
      }
      else {
            ychange = yin - yprev;
            ydir = 0;
      }
}

/*
Scale pixel values down to steps. The following will result in loss of
precision, but the PIC tracks its actual location to compensate.
*/
xSteps = xchange/5;
ySteps = ychange/5;


/*
Compensate for Etch-a-Sketch mechanical slippage.
*/
if(xdir != xdirprev)
      xStepsComp = xSteps + 2;
else
      xStepsComp = xSteps;
if(ydir != ydirprev)
      yStepsComp = ySteps + 1;
else
      yStepsComp = ySteps;

/*
Ensure that FPGA is ready to receive its next set of instructions.
*/
while(PORTCbits.RC0 == 0){};

/*
      1. Set reset high
      2. Load parallel I/O with new values
      3. Drop reset, instructing FPGA to draw
*/
PORTCbits.RC1 = 1;
PORTCbits.RC2 = xdir;
PORTCbits.RC3 = ydir;

// Pass both xStepsComp, yStepsComp at once
PORTD = (xStepsComp << 4) | (yStepsComp);
PORTCbits.RC1 = 0;

/* Store previous step parameters */
xdirprev = xdir;
ydirprev = ydir;

// Update xprev based on current xdir
if (xdir)
      xprev = xprev - (xSteps*5);
else
      xprev = xprev + (xSteps*5);
```

```
// Update yprev based on current ydir
      if (ydir)
            yprev = yprev - (ySteps*5);
      else
            yprev = yprev + (ySteps*5);

}


/*
----------------------
CHARACTER DEFINITIONS
----------------------

Characters defined as a sequence of line segments, drawn relative
to a reference point (xcurr, ycurr), situated at the lower left
corner of the character's "frame" (60x100 box).

*/


/* A - 60 x 100
*/
void drawA(void)
{
      drawpt(xcurr,ycurr+50);
      drawpt(xcurr,ycurr+100);

      drawpt(xcurr+60,ycurr+100);
      drawpt(xcurr+60,ycurr+50);
      drawpt(xcurr+60,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+40,ycurr+50);
      drawpt(xcurr+40,ycurr+80);
      drawpt(xcurr+20,ycurr+80);
      drawpt(xcurr+20,ycurr+60);
      drawpt(xcurr+40,ycurr+60);
      drawpt(xcurr+40,ycurr+40);
      drawpt(xcurr+20,ycurr+40);
      drawpt(xcurr+20,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}

/* B - 60 x 100
*/
void drawB(void)
{
      drawpt(xcurr,ycurr+50);
      drawpt(xcurr,ycurr+100);
      drawpt(xcurr+40,ycurr+100);
      drawpt(xcurr+60,ycurr+75);
      drawpt(xcurr+40,ycurr+50);
      drawpt(xcurr+60,ycurr+25);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+30,ycurr);
```

```
      drawpt(xcurr+30,ycurr+50);
      drawpt(xcurr+30,ycurr+80);
      drawpt(xcurr+20,ycurr+80);
      drawpt(xcurr+20,ycurr+60);
      drawpt(xcurr+30,ycurr+60);
      drawpt(xcurr+30,ycurr+40);
      drawpt(xcurr+20,ycurr+40);
      drawpt(xcurr+20,ycurr+20);
      drawpt(xcurr+30,ycurr+20);
      drawpt(xcurr+30,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}

/* C - 60 x 100
*/
void drawC(void)
{
      drawpt(xcurr,ycurr+50);
      drawpt(xcurr,ycurr+100);
      drawpt(xcurr+60,ycurr+100);
      drawpt(xcurr+60,ycurr+80);
      drawpt(xcurr+20,ycurr+80);
      drawpt(xcurr+20,ycurr+20);
      drawpt(xcurr+60,ycurr+20);
      drawpt(xcurr+60,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}

/* D - 60 x 100
*/
void drawD(void)
{
      drawpt(xcurr,ycurr+50);
      drawpt(xcurr,ycurr+100);
      drawpt(xcurr+40,ycurr+100);
      drawpt(xcurr+60,ycurr+80);
      drawpt(xcurr+60,ycurr+50);
      drawpt(xcurr+60,ycurr+20);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+40,ycurr+20);
      drawpt(xcurr+40,ycurr+50);
      drawpt(xcurr+40,ycurr+80);
      drawpt(xcurr+20,ycurr+80);
      drawpt(xcurr+20,ycurr+50);
      drawpt(xcurr+20,ycurr+20);
      drawpt(xcurr+40,ycurr+20);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}
```

```
/* E - 60 x 100
*/
void drawE(void)
{
      drawpt(xcurr,ycurr+50);
      drawpt(xcurr,ycurr+100);
      drawpt(xcurr+60,ycurr+100);
      drawpt(xcurr+60,ycurr+80);
      drawpt(xcurr+20,ycurr+80);
      drawpt(xcurr+20,ycurr+60);
      drawpt(xcurr+60,ycurr+60);
      drawpt(xcurr+60,ycurr+40);
      drawpt(xcurr+20,ycurr+40);
      drawpt(xcurr+20,ycurr+20);
      drawpt(xcurr+60,ycurr+20);
      drawpt(xcurr+60,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}

/* F - 60 x 100
*/
void drawF(void)
{
      drawpt(xcurr,ycurr+50);
      drawpt(xcurr,ycurr+100);
      drawpt(xcurr+60,ycurr+100);
      drawpt(xcurr+60,ycurr+80);
      drawpt(xcurr+20,ycurr+80);
      drawpt(xcurr+20,ycurr+60);
      drawpt(xcurr+60,ycurr+60);
      drawpt(xcurr+60,ycurr+40);
      drawpt(xcurr+20,ycurr+40);
      drawpt(xcurr+20,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}

/* G - 60 x 100
*/
void drawG(void)
{
      drawpt(xcurr,ycurr+50);
      drawpt(xcurr,ycurr+100);
      drawpt(xcurr+60,ycurr+100);
      drawpt(xcurr+60,ycurr+80);
      drawpt(xcurr+20,ycurr+80);
      drawpt(xcurr+20,ycurr+50);
      drawpt(xcurr+20,ycurr+20);
      drawpt(xcurr+40,ycurr+20);
      drawpt(xcurr+40,ycurr+40);
      drawpt(xcurr+30,ycurr+40);
      drawpt(xcurr+30,ycurr+60);
      drawpt(xcurr+60,ycurr+60);
      drawpt(xcurr+60,ycurr);
```

```
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}

/* H - 60 x 100
*/
void drawH(void)
{
      drawpt(xcurr,ycurr+50);
      drawpt(xcurr,ycurr+100);
      drawpt(xcurr+20,ycurr+100);
      drawpt(xcurr+20,ycurr+60);
      drawpt(xcurr+40,ycurr+60);
      drawpt(xcurr+40,ycurr+100);
      drawpt(xcurr+60,ycurr+100);
      drawpt(xcurr+60,ycurr+50);
      drawpt(xcurr+60,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+40,ycurr+40);
      drawpt(xcurr+20,ycurr+40);
      drawpt(xcurr+20,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}

/* I - 60 x 100
*/
void drawI(void)
{
      drawpt(xcurr,ycurr+20);
      drawpt(xcurr+20,ycurr+20);
      drawpt(xcurr+20,ycurr+80);
      drawpt(xcurr,ycurr+80);
      drawpt(xcurr,ycurr+100);
      drawpt(xcurr+60,ycurr+100);
      drawpt(xcurr+60,ycurr+80);
      drawpt(xcurr+40,ycurr+80);
      drawpt(xcurr+40,ycurr+20);
      drawpt(xcurr+60,ycurr+20);
      drawpt(xcurr+60,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}

/* J - 60 x 100
*/
void drawJ(void)
{
      drawpt(xcurr,ycurr+20);
      drawpt(xcurr+20,ycurr+20);
      drawpt(xcurr+20,ycurr+80);
      drawpt(xcurr,ycurr+80);
      drawpt(xcurr,ycurr+100);
      drawpt(xcurr+60,ycurr+100);
```

```
      drawpt(xcurr+60,ycurr+80);
      drawpt(xcurr+40,ycurr+80);
      drawpt(xcurr+40,ycurr+50);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}

/* K - 60 x 100
*/
void drawK(void)
{
      drawpt(xcurr,ycurr+50);
      drawpt(xcurr,ycurr+100);
      drawpt(xcurr+20,ycurr+100);
      drawpt(xcurr+20,ycurr+60);
      drawpt(xcurr+40,ycurr+80);
      drawpt(xcurr+40,ycurr+100);
      drawpt(xcurr+60,ycurr+100);
      drawpt(xcurr+60,ycurr+80);
      drawpt(xcurr+40,ycurr+60);
      drawpt(xcurr+30,ycurr+50);
      drawpt(xcurr+40,ycurr+40);
      drawpt(xcurr+60,ycurr+20);
      drawpt(xcurr+60,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+40,ycurr+20);
      drawpt(xcurr+20,ycurr+40);
      drawpt(xcurr+20,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}

/* L - 60 x 100
*/
void drawL(void)
{
      drawpt(xcurr,ycurr+50);
      drawpt(xcurr,ycurr+100);
      drawpt(xcurr+20,ycurr+100);
      drawpt(xcurr+20,ycurr+50);
      drawpt(xcurr+20,ycurr+20);
      drawpt(xcurr+60,ycurr+20);
      drawpt(xcurr+60,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}

/* M - 60 x 100
*/
void drawM(void)
{
      drawpt(xcurr,ycurr+50);
      drawpt(xcurr,ycurr+100);
```

```
      drawpt(xcurr+20,ycurr+100);
      drawpt(xcurr+30,ycurr+80);
      drawpt(xcurr+40,ycurr+100);
      drawpt(xcurr+60,ycurr+100);
      drawpt(xcurr+60,ycurr+50);
      drawpt(xcurr+60,ycurr);
      drawpt(xcurr+50,ycurr);
      drawpt(xcurr+50,ycurr+50);
      drawpt(xcurr+50,ycurr+80);
      drawpt(xcurr+30,ycurr+60);
      drawpt(xcurr+10,ycurr+80);
      drawpt(xcurr+10,ycurr+50);
      drawpt(xcurr+10,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}

/* N - 60 x 100
*/
void drawN(void)
{
      drawpt(xcurr,ycurr+50);
      drawpt(xcurr,ycurr+100);
      drawpt(xcurr+20,ycurr+100);
      drawpt(xcurr+30,ycurr+60);
      drawpt(xcurr+50,ycurr+20);
      drawpt(xcurr+50,ycurr+50);
      drawpt(xcurr+50,ycurr+100);
      drawpt(xcurr+60,ycurr+100);
      drawpt(xcurr+60,ycurr+50);
      drawpt(xcurr+60,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+30,ycurr+40);
      drawpt(xcurr+10,ycurr+80);
      drawpt(xcurr+10,ycurr+50);
      drawpt(xcurr+10,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}

/* O - 60 x 100
*/
void drawO(void)
{
      drawpt(xcurr,ycurr+50);
      drawpt(xcurr,ycurr+100);
      drawpt(xcurr+60,ycurr+100);
      drawpt(xcurr+60,ycurr+50);
      drawpt(xcurr+60,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+40,ycurr+50);
      drawpt(xcurr+40,ycurr+80);
      drawpt(xcurr+20,ycurr+80);
      drawpt(xcurr+20,ycurr+20);
      drawpt(xcurr+40,ycurr+20);
```

```
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}

/* P - 60 x 100
*/
void drawP(void)
{
      drawpt(xcurr,ycurr+50);
      drawpt(xcurr,ycurr+100);
      drawpt(xcurr+60,ycurr+100);
      drawpt(xcurr+60,ycurr+40);
      drawpt(xcurr+20,ycurr+40);
      drawpt(xcurr+20,ycurr+80);
      drawpt(xcurr+40,ycurr+80);
      drawpt(xcurr+40,ycurr+60);
      drawpt(xcurr+20,ycurr+60);
      drawpt(xcurr+20,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}

/* Q - 60 x 100
*/
void drawQ(void)
{
      drawpt(xcurr+20,ycurr);
      drawpt(xcurr,ycurr+20);
      drawpt(xcurr,ycurr+80);
      drawpt(xcurr+20,ycurr+100);
      drawpt(xcurr+40,ycurr+100);
      drawpt(xcurr+60,ycurr+80);
      drawpt(xcurr+60,ycurr+20);
      drawpt(xcurr+55,ycurr+15);
      drawpt(xcurr+60,ycurr+15);
      drawpt(xcurr+45,ycurr);
      drawpt(xcurr+45,ycurr+5);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+40,ycurr+50);
      drawpt(xcurr+40,ycurr+80);
      drawpt(xcurr+20,ycurr+80);
      drawpt(xcurr+20,ycurr+20);
      drawpt(xcurr+40,ycurr+20);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}
```

```
/* R - 60 x 100
*/
void drawR(void)
{
      drawpt(xcurr,ycurr+50);
      drawpt(xcurr,ycurr+100);
      drawpt(xcurr+60,ycurr+100);
      drawpt(xcurr+60,ycurr+50);
      drawpt(xcurr+60,ycurr+40);
      drawpt(xcurr+40,ycurr+40);
      drawpt(xcurr+60,ycurr);
      drawpt(xcurr+50,ycurr);
      drawpt(xcurr+20,ycurr+40);
      drawpt(xcurr+20,ycurr+80);
      drawpt(xcurr+40,ycurr+80);
      drawpt(xcurr+40,ycurr+60);
      drawpt(xcurr+20,ycurr+60);
      drawpt(xcurr+20,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}

/* S - 60 x 100
*/
void drawS(void)
{
      drawpt(xcurr,ycurr+20);
      drawpt(xcurr+40,ycurr+20);
      drawpt(xcurr+40,ycurr+40);
      drawpt(xcurr+20,ycurr+40);
      drawpt(xcurr,ycurr+60);
      drawpt(xcurr,ycurr+80);
      drawpt(xcurr+20,ycurr+100);
      drawpt(xcurr+60,ycurr+100);
      drawpt(xcurr+60,ycurr+80);
      drawpt(xcurr+20,ycurr+80);
      drawpt(xcurr+20,ycurr+60);
      drawpt(xcurr+40,ycurr+60);
      drawpt(xcurr+60,ycurr+40);
      drawpt(xcurr+60,ycurr+20);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}

/* T - 60 x 100
*/
void drawT(void)
{
      drawpt(xcurr+20,ycurr);
      drawpt(xcurr+20,ycurr+50);
      drawpt(xcurr+20,ycurr+80);
      drawpt(xcurr,ycurr+80);
      drawpt(xcurr,ycurr+100);
      drawpt(xcurr+60,ycurr+100);
```

```
      drawpt(xcurr+60,ycurr+80);
      drawpt(xcurr+40,ycurr+80);
      drawpt(xcurr+40,ycurr+50);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}

/* U - 60 x 100
*/
void drawU(void)
{
      drawpt(xcurr,ycurr+50);
      drawpt(xcurr,ycurr+100);
      drawpt(xcurr+20,ycurr+100);
      drawpt(xcurr+20,ycurr+50);
      drawpt(xcurr+20,ycurr+20);
      drawpt(xcurr+40,ycurr+20);
      drawpt(xcurr+40,ycurr+50);
      drawpt(xcurr+40,ycurr+100);
      drawpt(xcurr+60,ycurr+100);
      drawpt(xcurr+60,ycurr+50);
      drawpt(xcurr+60,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}

/* V - 60 x 100
*/
void drawV(void)
{
      drawpt(xcurr+20,ycurr);
      drawpt(xcurr+10,ycurr+20);
      drawpt(xcurr,ycurr+40);
      drawpt(xcurr,ycurr+100);
      drawpt(xcurr+20,ycurr+100);
      drawpt(xcurr+20,ycurr+60);
      drawpt(xcurr+30,ycurr+20);
      drawpt(xcurr+40,ycurr+60);
      drawpt(xcurr+40,ycurr+100);
      drawpt(xcurr+60,ycurr+100);
      drawpt(xcurr+60,ycurr+40);
      drawpt(xcurr+50,ycurr+20);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}

/* W - 60 x 100
*/
void drawW(void)
{
      drawpt(xcurr,ycurr+50);
      drawpt(xcurr,ycurr+100);
```

```
	drawpt(xcurr+20,ycurr+100);
	drawpt(xcurr+20,ycurr+50);
	drawpt(xcurr+20,ycurr+20);
	drawpt(xcurr+30,ycurr+40);
	drawpt(xcurr+40,ycurr+20);
	drawpt(xcurr+40,ycurr+50);
	drawpt(xcurr+40,ycurr+100);
	drawpt(xcurr+60,ycurr+100);
	drawpt(xcurr+60,ycurr+50);
	drawpt(xcurr+60,ycurr);
	drawpt(xcurr+40,ycurr);
	drawpt(xcurr+30,ycurr+20);
	drawpt(xcurr+20,ycurr);
	drawpt(xcurr,ycurr);
	drawpt(xcurr+40,ycurr);
	drawpt(xcurr+80,ycurr);
}

/* X - 60 x 100
*/
void drawX(void)
{
	drawpt(xcurr,ycurr+25);
	drawpt(xcurr+20,ycurr+50);
	drawpt(xcurr,ycurr+75);
	drawpt(xcurr,ycurr+100);
	drawpt(xcurr+20,ycurr+100);
	drawpt(xcurr+20,ycurr+75);
	drawpt(xcurr+30,ycurr+60);
	drawpt(xcurr+40,ycurr+75);
	drawpt(xcurr+40,ycurr+100);
	drawpt(xcurr+60,ycurr+100);
	drawpt(xcurr+60,ycurr+75);
	drawpt(xcurr+40,ycurr+50);
	drawpt(xcurr+60,ycurr+25);
	drawpt(xcurr+60,ycurr);
	drawpt(xcurr+40,ycurr);
	drawpt(xcurr+40,ycurr+25);
	drawpt(xcurr+30,ycurr+40);
	drawpt(xcurr+20,ycurr+25);
	drawpt(xcurr+20,ycurr);
	drawpt(xcurr,ycurr);
	drawpt(xcurr+40,ycurr);
	drawpt(xcurr+80,ycurr);
}

/* Y - 60 x 100
*/
void drawY(void)
{
	drawpt(xcurr+20,ycurr);
	drawpt(xcurr+20,ycurr+50);
	drawpt(xcurr,ycurr+75);
	drawpt(xcurr,ycurr+100);
	drawpt(xcurr+20,ycurr+100);
	drawpt(xcurr+20,ycurr+70);
	drawpt(xcurr+40,ycurr+70);
```

```
      drawpt(xcurr+40,ycurr+100);
      drawpt(xcurr+60,ycurr+100);
      drawpt(xcurr+60,ycurr+75);
      drawpt(xcurr+40,ycurr+50);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}

/* Z - 60 x 100
*/
void drawZ(void)
{
      drawpt(xcurr,ycurr+20);
      drawpt(xcurr+40,ycurr+80);
      drawpt(xcurr,ycurr+80);
      drawpt(xcurr,ycurr+100);
      drawpt(xcurr+60,ycurr+100);
      drawpt(xcurr+60,ycurr+80);
      drawpt(xcurr+20,ycurr+20);
      drawpt(xcurr+60,ycurr+20);
      drawpt(xcurr+60,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}

/* 1 - 60 x 100
*/
void draw1(void)
{
      drawpt(xcurr+20,ycurr);
      drawpt(xcurr+20,ycurr+50);
      drawpt(xcurr+20,ycurr+100);
      drawpt(xcurr+40,ycurr+100);
      drawpt(xcurr+40,ycurr+50);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}

/* 2 - 60 x 100
*/
void draw2(void)
{
      drawpt(xcurr,ycurr+60);
      drawpt(xcurr+40,ycurr+60);
      drawpt(xcurr+40,ycurr+80);
      drawpt(xcurr,ycurr+80);
      drawpt(xcurr,ycurr+100);
      drawpt(xcurr+60,ycurr+100);
      drawpt(xcurr+60,ycurr+40);
      drawpt(xcurr+20,ycurr+40);
      drawpt(xcurr+20,ycurr+20);
      drawpt(xcurr+60,ycurr+20);
```

```
	drawpt(xcurr+60,ycurr);
	drawpt(xcurr,ycurr);
	drawpt(xcurr+40,ycurr);
	drawpt(xcurr+80,ycurr);
}

/* 3 - 60 x 100
*/
void draw3(void)
{
	drawpt(xcurr,ycurr+20);
	drawpt(xcurr+40,ycurr+20);
	drawpt(xcurr+40,ycurr+40);
	drawpt(xcurr,ycurr+40);
	drawpt(xcurr,ycurr+60);
	drawpt(xcurr+40,ycurr+60);
	drawpt(xcurr+40,ycurr+80);
	drawpt(xcurr,ycurr+80);
	drawpt(xcurr,ycurr+100);
	drawpt(xcurr+60,ycurr+100);
	drawpt(xcurr+60,ycurr+50);
	drawpt(xcurr+60,ycurr);
	drawpt(xcurr,ycurr);
	drawpt(xcurr+40,ycurr);
	drawpt(xcurr+80,ycurr);
}

/* 4 - 60 x 100
*/
void draw4(void)
{
	drawpt(xcurr+40,ycurr);
	drawpt(xcurr+40,ycurr+40);
	drawpt(xcurr,ycurr+40);
	drawpt(xcurr,ycurr+100);
	drawpt(xcurr+20,ycurr+100);
	drawpt(xcurr+20,ycurr+60);
	drawpt(xcurr+40,ycurr+60);
	drawpt(xcurr+40,ycurr+100);
	drawpt(xcurr+60,ycurr+100);
	drawpt(xcurr+60,ycurr+50);
	drawpt(xcurr+60,ycurr);
	drawpt(xcurr,ycurr);
	drawpt(xcurr+40,ycurr);
	drawpt(xcurr+80,ycurr);
}

/* 5 - 60 x 100
*/
void draw5(void)
{
	drawpt(xcurr,ycurr+20);
	drawpt(xcurr+40,ycurr+20);
	drawpt(xcurr+40,ycurr+40);
	drawpt(xcurr,ycurr+40);
	drawpt(xcurr,ycurr+100);
	drawpt(xcurr+60,ycurr+100);
```

```
      drawpt(xcurr+60,ycurr+80);
      drawpt(xcurr+20,ycurr+80);
      drawpt(xcurr+20,ycurr+60);
      drawpt(xcurr+60,ycurr+60);
      drawpt(xcurr+60,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}

/* 6 - 60 x 100
*/
void draw6(void)
{
      drawpt(xcurr,ycurr+50);
      drawpt(xcurr,ycurr+100);
      drawpt(xcurr+60,ycurr+100);
      drawpt(xcurr+60,ycurr+80);
      drawpt(xcurr+20,ycurr+80);
      drawpt(xcurr+20,ycurr+60);
      drawpt(xcurr+60,ycurr+60);
      drawpt(xcurr+60,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+40,ycurr+40);
      drawpt(xcurr+20,ycurr+40);
      drawpt(xcurr+20,ycurr+20);
      drawpt(xcurr+40,ycurr+20);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}

/* 7 - 60 x 100
*/
void draw7(void)
{
      drawpt(xcurr+20,ycurr);
      drawpt(xcurr+30,ycurr+40);
      drawpt(xcurr+40,ycurr+80);
      drawpt(xcurr,ycurr+80);
      drawpt(xcurr,ycurr+100);
      drawpt(xcurr+60,ycurr+100);
      drawpt(xcurr+60,ycurr+80);
      drawpt(xcurr+50,ycurr+40);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}
```

```
/* 8 - 60 X 100
*/
void draw8(void)
{
      drawpt(xcurr,ycurr+50);
      drawpt(xcurr,ycurr+100);
      drawpt(xcurr+60,ycurr+100);
      drawpt(xcurr+60,ycurr+50);
      drawpt(xcurr+60,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+40,ycurr+50);
      drawpt(xcurr+40,ycurr+80);
      drawpt(xcurr+20,ycurr+80);
      drawpt(xcurr+20,ycurr+60);
      drawpt(xcurr+40,ycurr+60);
      drawpt(xcurr+40,ycurr+40);
      drawpt(xcurr+20,ycurr+40);
      drawpt(xcurr+20,ycurr+20);
      drawpt(xcurr+40,ycurr+20);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}

/* 9 - 60 X 100
*/
void draw9(void)
{
      drawpt(xcurr,ycurr+20);
      drawpt(xcurr+40,ycurr+20);
      drawpt(xcurr+40,ycurr+80);
      drawpt(xcurr+20,ycurr+80);
      drawpt(xcurr+20,ycurr+60);
      drawpt(xcurr+40,ycurr+60);
      drawpt(xcurr+40,ycurr+40);
      drawpt(xcurr,ycurr+40);
      drawpt(xcurr,ycurr+100);
      drawpt(xcurr+60,ycurr+100);
      drawpt(xcurr+60,ycurr+50);
      drawpt(xcurr+60,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}

/* 0 - 60 x 100
*/
void draw0(void)
{
      drawpt(xcurr+20,ycurr);
      drawpt(xcurr,ycurr+20);
      drawpt(xcurr,ycurr+80);
      drawpt(xcurr+20,ycurr+100);
      drawpt(xcurr+40,ycurr+100);
      drawpt(xcurr+60,ycurr+80);
      drawpt(xcurr+60,ycurr+20);
```

```
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+40,ycurr+50);
      drawpt(xcurr+40,ycurr+80);
      drawpt(xcurr+20,ycurr+80);
      drawpt(xcurr+20,ycurr+20);
      drawpt(xcurr+40,ycurr+20);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr,ycurr);
      drawpt(xcurr+40,ycurr);
      drawpt(xcurr+80,ycurr);
}

/* ' ' - 60 x 100
*/
void drawSpace(void)
{
      drawpt(xcurr + 40, ycurr);
      drawpt(xcurr + 80, ycurr);
}
```

Appendix C: FPGA Verilog Code

```
`timescale 1ns / 1ps

/*
Verilog Module: motor_driver

Authors: dbujalski, jkarras@hmc.edu
Created: November 2009

High level Verilog module for the stepper motor
driver. Links a clock divider with two step counters
and two motor FSMs (vertical and horizontal controls).

*/
module motor_driver( input          clk, reset,
                     input    [3:0] horiz_steps, vert_steps,
                     input          horiz_dir, vert_dir,
                     output         ready,
                     output         fsm_horiz_en, fsm_vert_en,
                     output   [3:0] horiz_motor,vert_motor);

      wire slow_clk;

      // Divide system clock (20 MHz) down to about 10 Hz
      clock_divider #(21) clk_div (clk, reset, slow_clk);

      // Two step counters to enable the FSMs driving the motors
      step_counter ctr_horiz (slow_clk, reset, horiz_steps, fsm_horiz_en);
      step_counter ctr_vert (slow_clk, reset, vert_steps, fsm_vert_en);

      // FPGA is ready as soon as both motor enables are low
      assign ready = ~(fsm_horiz_en || fsm_vert_en);

      // Motor FSMs to track state of steppers, adjust motor phases
      motor_fsm run_horiz_motor (slow_clk, fsm_horiz_en, horiz_dir,
                                 horiz_motor);
      motor_fsm run_vert_motor (slow_clk, fsm_vert_en, vert_dir, vert_motor);

endmodule
```

```
`timescale 1ns / 1ps

/*
Verilog Module: clock_divider

A simple clock divider, based on an N-bit counter.
The bitwidth of the counter is parametrized, with
a default value of 4 bits. The output clock is
simply tied to the MSB of the counter.

Author: Jaakko Karras, jkarras@hmc.edu
Created: November 13, 2009
*/

module clock_divider #( parameter N = 4 )
                       ( input       clk, reset,
                         output      div_clk );

      reg [N-1:0] count = 0;

      /*
      Counter Block: Counter will count up from zero, overflow, and
      start again.
      */
      always @ (posedge clk, posedge reset)
            if (reset) count <= 0;
            else count <= count + 1;

      /*
      Output Logic
      */
      assign div_clk = count[N-1];


endmodule
```

```
`timescale 1ns / 1ps

/*
Verilog Module: step_counter

Authors: dbujalski, jkarras@hmc.edu
Created: November 2009

A module for counting through the specified number
of stepper motor "cycles". The PIC specifices a certain
number of steps (0 - 15) on both knobs, which the FPGA
then carries out. The step counter maintains an
enable signal, which is kept high until the target number
of cycles is reached. The enable signal drives the FSM
corresponding to the step counter.
*/
module step_counter( input          clk, reset,
                     input    [3:0] target,
                     output         en_fsm );

      reg [3:0] count;
      reg en;

      // Counter to count up to the target number of
      // cycles
      always @ ( posedge clk, posedge reset )
            if (reset)        count <= 0;
            else if (en)      count <= count + 1;


      // Output (enable signal) logic
      always @ ( * )
            if (count == target)          en = 0;
            else                          en = 1;

      // Assign output signal
      assign en_fsm = en;

endmodule
```

```
`timescale 1ns / 1ps

/*
Verilog Module: motor_fsm

Authors: dbujalski, jkarras@hmc.edu
Created: November 2009

Actuates a stepper motor by "stepping" through
each of the four possible states, in the order
specified by the direction input. The FSM requires
that the enable signal be high in order for it
to function.

DIRECTION CONVENTION:
0 = Turn counter clockwise
1 = Turn clockwise
*/

module motor_fsm(input        clk, en, dir,
                 output [3:0] motor);

      /*
      bit 3 - h_1 right on
      bit 2 - h_1 left on
      bit 1 - h_2 right on
      bit 0 - h_2 left on
      */
      parameter h1_L_h2_L = 4'b0101;
      parameter h1_L_h2_R = 4'b0110;
      parameter h1_R_h2_R = 4'b1010;
      parameter h1_R_h2_L = 4'b1001;


      reg [3:0] state, nextstate;


      // Transition Logic
      always @ ( posedge clk )
            if (en)           state <= nextstate;


      // Next State Logic
      always @ ( * )
            case (state)
                  h1_L_h2_L:  begin
                              if (dir)    //Shift by direction
                                    nextstate = h1_L_h2_R;
                              else
                                    nextstate = h1_R_h2_L;
                              end
                  h1_L_h2_R:  begin
                              if (dir)
                                    nextstate = h1_R_h2_R;
                              else
                                    nextstate = h1_L_h2_L;
                              end
```

```
            h1_R_h2_R:  begin
                        if (dir)
                              nextstate = h1_R_h2_L;
                        else
                              nextstate = h1_L_h2_R;
                        end
            h1_R_h2_L:  begin
                        if (dir)
                              nextstate = h1_L_h2_L;
                        else
                              nextstate = h1_R_h2_R;
                        end
            default:    nextstate = h1_L_h2_L;
      endcase

      //Output logic
      assign motor = state;

endmodule
```