

For each of these exercises, prepare a written description and/or circuit diagram and/or code listing (as appropriate) explaining what you did and how you came to believe your interpretation of what was going on was correct.

1. Experiment with the Handy Cricket and the Cricket Logo software. Play with motor and sensor commands. Display sensor data back on the the Cricket Logo screen.

We connected a photocell to the sensor A input and a 9V LEGO motor to the motor A output. Then we ran the following commands:

`beep` → The Cricket beeped.

`repeat 10 [beep]` → The Cricket beeped 10 times.

`note 62 50` → The Cricket played “b” note for 5 seconds.

`a, onfor 20` → The motor ran forward for 2 seconds.

`a, thatway onfor 20` → The motor ran backward for 2 seconds.

`a, thisway onfor 20` → The motor ran forward for 2 seconds.

`a, repeat 5 [rd onfor 20]` → The motor ran backward, forward, backward, forward, backward for 2 sec each way.

`send 3 * 4` → The monitor displayed “24”.

`loop [send sensora]` → The monitor continually displayed photocell values updated every $\frac{1}{10}$ second. The values ranged from: 2 (using a flashlight), 33 (ambient room light), 231 (with my thumb covering it), 255 (with my hand covering it).

`when [newir?] [stop!] loop [beep wait 10]` → The Cricket beeped every second until we transmitted data to it from the PC. To do that, we simply attempted to send a command from the command center. The send failed (by displaying the “No Cricket in sight” dialog box) because the Cricket was not in the monitor, it did not complete the normal protocol.

¹I worked with Roger Matar in the lab for most of these problems.

2. Experiment with the Cricket's IR communications primitives `send`, `newir?`, and `ir`. Get one Cricket to trigger another Cricket to do something.

We attached the photocell to the sensor A input of a Cricket running the program
`when [switcha] [send 4]`.

We attached a motor to the motor A output to a second Cricket running the program
`when [newir?] [a, onfor ir * 10]`.

When we covered the photocell so that its value would be higher than the switch threshold value. That caused the first Cricket to evaluate “switcha” to true and so send the value “4” over its IR port.

The second Cricket received the value, which caused “newir?” to return true. As a result, it ran motor A for the number of seconds sent by the first Cricket (i.e. 4 seconds).

3. Create a multi-Cricket application whereby the Crickets pass a software token from one to another (“multi” ≥ 2). When a Cricket has the token, it should display some behavior to make this evident (e.g., beeping or turning on a motor output). Then this Cricket should be able to pass the token to another Cricket. Are there any problems in accomplishing this? Are the problems of a fundamental nature, or just implementation details?

Our first attempt used the “pair of state machines that clocked each other back and forth,” as you described in your e-mail message. Each cricket ran this program:

```
loop [
  repeat 5 [ beep wait 10 ]
  send 4
  waituntil [ newir? and (ir = 4) ]
]
```

The only interesting aspect here was that we added the “and ir=4” clause to filter stray infrared data from the neighboring bench.

I started over when I received your “multi is >2” e-mail message.

First, I prototyped using one Cricket and the PC. I wrote a simple Java program to simulate half of the protocol and monitored how the Cricket responded. I found it was useful to have the Cricket send values in the range 0xF0-0xFF to debug the program flow.

When I was confident that the protocol worked, I loaded the program into three Crickets... and my confidence was shattered. The system was completely unreliable. Sometimes I had two Crickets claiming to hold the token and sometimes all of them were requesting the token². The most significant change from that debugging session was to have the program repeatedly send the “give token” message until it received a “have token” message from its intended destination.

²The lesson here is that nothing beats the real thing.

The final program uses single-byte messages where the low nibble contains the Cricket ID and the high nibble is the operation: $0x1n$ for “ n has the token”, $0x2n$ for “ n wants the token”, $0x3n$ for “give the token to n ”.

For example the sequence to transfer the token from C1 to C2 is:

- (1) C1 indicates it has the token by sending $0x11$ (have-1).
- (2) C2 requests the token by sending $0x22$ (want-2).
- (3) When C1 receives $0x22$ (want-2), sends $0x32$ (give-2).
- (4) When C2 receives $0x32$ (give-2), it beeps five times (for our benefit), sets `haveit true` and sends $0x12$ (have-2).
- (5) When C1 receives $0x12$ (have-2), it sets `haveit false`.

I used the same program on each of the Crickets, although I modified the “setme” line before each download³.

```
main
global [me haveit recv]

to giveit :target
  loop [
    send 0x30 or :target
    if newir? and (ir = (0x10 or :target)) [ sethaveit 0 stop ]
  ]
end

to grabit
  sethaveit 1
  repeat 5 [ wait 6 beep ]
end

to main
  setme 1
  sethaveit (me = 1)
  loop [
    ifelse haveit [
      send 0x10 or me
      beep
      if newir? [
        setrecv ir
        if (recv and 0xF0) = 0x20 [ giveit recv and 0x0F ]
      ]
      wait 10
    ] [
      send 0x20 or me
      if newir? and (ir = (0x30 or me)) [ grabit ]
      wait 15
    ]
  ]
end
```

³A better approach would have been to store the ID outside the program and then retrieve it at startup.

4. Devise an experiment to determine if Cricket Logo arrays use 0 or 1 as an index for the first array element. (The Cricket Logo documentation does not specify.) You may wish to refer to the Cricket memory map at <http://handyboard.com/cricket/tech/>. See also the Cricket Logo commands “examine byte” (eb) and “deposit byte” (db).

The memory map at <http://www.handyboard.com/cricket/tech/memory.txt> shows that user data is stored at 0x000-0xfef and that user arrays are first in memory. So, we assumed that our first array would start at address 0x000. And since numbers are two bytes long, we expected addresses 0x000-0x001 to store the value of the first element of the first array and 0x002-0x003 to store the second element of the first array.

We ran the following program on the cricket:

```
array [x 3 y 3]
to find
  aset x 0 0x0102
  aset x 1 0x0304
  send eb 0 beep wait 20
  send eb 1 beep wait 20
  send eb 2 beep wait 20
  send eb 3 beep
end
```

The monitor displayed 1, 2, 3, 4 with two second pauses between each value (and the cricket beeped as each value was displayed). Since the first “aset” command used an index of 0, and the first two bytes correspond to the values it stored, we can conclude arrays are zero-based.

To double-check the results, we extended the program to overrun and underrun the arrays:

```
array [x 2 y 2]
to find
  aset x 0 0x0102
  aset x 1 0x0304
  aset y 0 0x0506
  aset y 1 0x0708
  aset y -1 0x090A
  aset x 2 0x0B0C
  send eb 0 beep wait 20
  send eb 1 beep wait 20
  send eb 2 beep wait 20
  send eb 3 beep wait 20
  send eb 4 beep wait 20
  send eb 5 beep wait 20
  send eb 6 beep wait 20
  send eb 7 beep wait 20
end
```

The monitor displayed 1, 2, 9, 10, 11, 12, 7, 8 as we expected. The command “aset y -1” wrote to the last element of array x. The command “aset x 2” wrote to the first element of array y.

5. Put a program on the Cricket that continually transmits sensor values (e.g., `loop [send sensora]`). Write a program that runs on a conventional desktop/laptop computer or a PDA that reads these values off the serial port (open the serial port with settings 9600-N-8-1) and displays them in some visual fashion (e.g., the music visualizations made by your desktop MP3 player).

We attached a photocell to the sensor A input of a cricket running the program:
`loop [send sensora]`.

I tried using “fastsend” rather than “send”, but neither program could keep up with the data flow. As a result, the rendering would continue for a few seconds after I stopped the Cricket. The data buffering ruined the immediacy.

We decided to use Java to read the serial port of the PC. Since the Sun JRE does not include serial port support, we downloaded the Java Communications API⁴. Since the package has not been updated since November 1998 (JRE 1.1), the installation instructions were out of date. To install it into JRE 1.3 or newer, run these commands⁵ after unpacking the distribution:

```
cd /d C:\commapi
copy comm.jar \jdk1.3.1\jre\lib\ext
copy win32com.dll \jdk1.3.1\bin
copy javax.comm.properties \jdk1.3.1\jre\lib
```

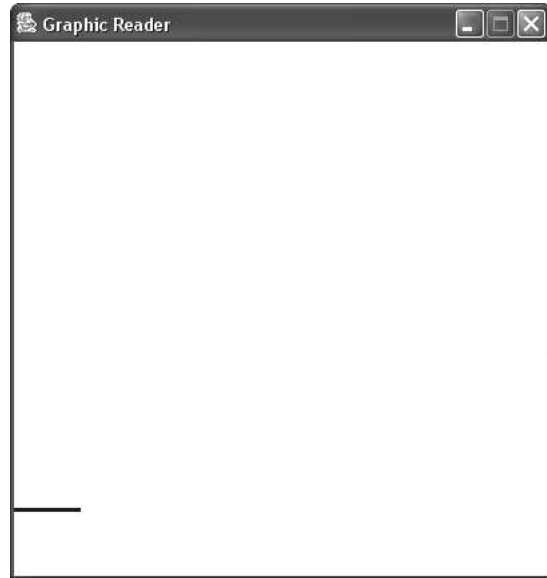
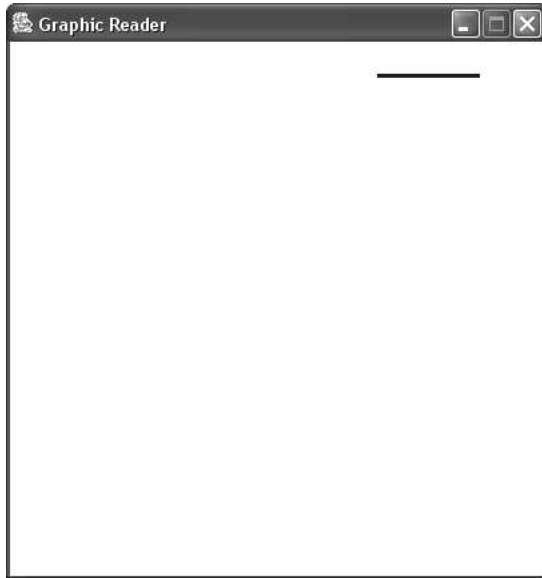
Our first attempt simply printed a variable-length line of dashes for each value it received (see page 7).

Once we had this working, I implemented a graphical version using the Java AWT (see page 8). For each value it receives from the Cricket, the program uses the low four bits for the X axis offset and the high four bits for the Y axis offset. The half-byte values are scaled to fill the size of the window (400 × 400 pixels). It maintains a buffer of 20 of these X-Y values and draws a line connecting the points each time it receives a new value.

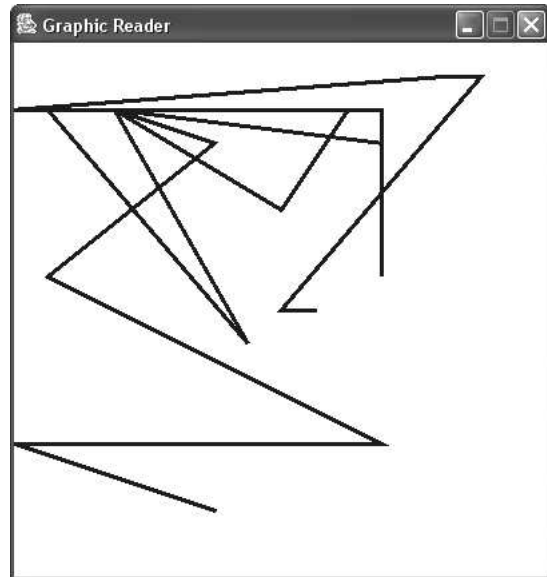
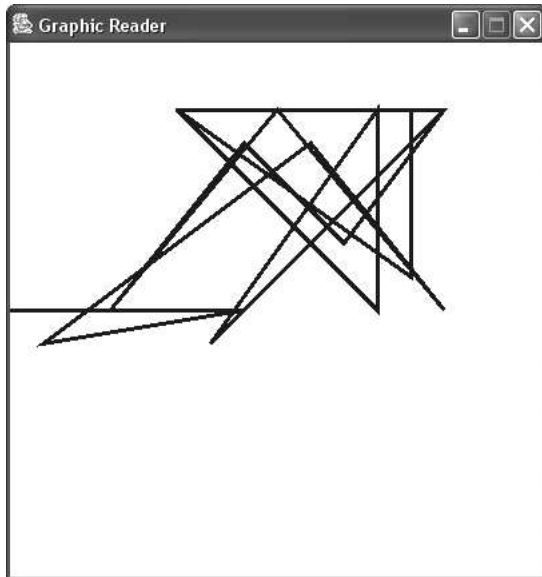
⁴See <http://java.sun.com/products/javacomm>

⁵From <http://developer.java.sun.com/developer/JDCTechTips/2002/tt0122.html#tip2>

These two examples show the well-lit steady state (left) and the dark steady state (right). In both cases, slight variation in the photocell value creates a horizontal line. Since the photocell value is inversely proportional to the amount of light, the line for the well-lit case is near the top of the window (the Y-axis zero) while the dark case is near the bottom (the maximum of the Y-axis):



We can also create patterns by moving the light sensor around:



My favorite effect is when I create a complex pattern and then stop the Cricket. The program quits receiving data and the pattern collapses down. Kinda cool.

The text-based Cricket reader (TextReader.java)

```
import java.io.InputStream;
import javax.comm.CommPortIdentifier;
import javax.comm.SerialPort;

/**
 * Read data from the HandyCricket and display it as a scrolling bar chart.
 * This code depends on the Java Communications API available at
 * http://java.sun.com/products/javacomm.
 */
public class TextReader                                     10
{
    public static void main(String[] args)
    {
        try {
            CommPortIdentifier id = CommPortIdentifier.getPortIdentifier("COM2");
            SerialPort port = (SerialPort) id.open("Reader", /*timeout=*/ 20000);
            port.setSerialPortParams(/*baudrate=*/ 9600,
                                     SerialPort.DATABITS_8,
                                     SerialPort.STOPBITS_1,
                                     SerialPort.PARITY_NONE);           20
            InputStream in = port.getInputStream();

            while (true) {
                int value = 80 - (in.read() * 80 / 256);
                while (value-- > 0)
                    System.out.print("-");
                System.out.println("");
            }

            } catch (Exception e) {                                     30
                e.printStackTrace(System.out);
            }
        }
    }
}
```

The GUI-based Cricket reader (GraphicReader.java)

```
import java.awt.*;
import java.awt.event.*;
import java.io.InputStream;
import java.util.Arrays;
import javax.comm.CommPortIdentifier;
import javax.comm.SerialPort;

public class GraphicReader extends Frame
{
    private static final int BUFFER_SIZE = 20;
    private static final int CANVAS_WIDTH = 400;
    private static final int CANVAS_HEIGHT = 400;

    public static void main(String[] args)
    {
        try {
            GraphicReader window = new GraphicReader();

            CommPortIdentifier id = CommPortIdentifier.getPortIdentifier("COM2");
            SerialPort port = (SerialPort) id.open("Reader", /*timeout=*/ 20000);
            port.setSerialPortParams(/*baudrate=*/ 9600,
                SerialPort.DATABITS_8,
                SerialPort.STOPBITS_1,
                SerialPort.PARITY_NONE);
            InputStream in = port.getInputStream();

            while (true)
                window.addValue(in.read());

        } catch (Exception e) {
            e.printStackTrace(System.out);
        }
    }

    private PlotCanvas fCanvas = new PlotCanvas();

    GraphicReader()
    {
        super("Graphic Reader");

        addWindowListener(new CloseHandler());
        setResizable(false);
        add(fCanvas);
        pack();
        show();
    }
}
```

```
void addValue(int value)
{
    int x = (value & 0x0F) * (CANVAS_WIDTH / 16);
    int y = ((value & 0xF0) >> 4) * (CANVAS_HEIGHT / 16);
    fCanvas.addPoint(x, y);
}

class PlotCanvas extends Canvas
{
    private int[] fPointX = new int[BUFFER_SIZE];
    private int[] fPointY = new int[BUFFER_SIZE];

    PlotCanvas()
    {
        setSize(CANVAS_WIDTH, CANVAS_HEIGHT);
    }

    void addPoint(int x, int y)
    {
        for (int index = 0; index < BUFFER_SIZE-1; index++) {
            fPointX[index] = fPointX[index+1];
            fPointY[index] = fPointY[index+1];
        }
        fPointX[BUFFER_SIZE-1] = x;
        fPointY[BUFFER_SIZE-1] = y;
        repaint();
    }

    public void paint(Graphics g)
    {
        ((Graphics2D) g).setStroke(new BasicStroke(3.0f));
        g.setColor(Color.blue);
        g.drawPolyline(fPointX, fPointY, BUFFER_SIZE);
    }
}

class CloseHandler extends WindowAdapter
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
}
}
```

6. If you get done with all of the above, then:

Read the information at <http://handyboard.com/cricket/tech/> regarding the Cricket's byte code language and communications protocol. Write an application that interacts with the Cricket and causes it to turn on its motor port. The sequence of steps is as follows:

1. Open the serial port for 9600-N-8-1 communications.
2. Go through the cricket-check sequence and retrieve the decimal 135 acknowledgment value. Make sure to trap and discard the hardware and software echoes.
3. Use the set-pointer command to point to an arbitrary address in memory.
4. Load in the code for a, on.
5. Set the pointer back to the initial location of the code.
6. Send the run command.

I did not have time to complete this problem. But I started it by building CricketComm, a Java class that manages communication with the Cricket. I used this class for the problem 3 prototyping. The next step would be to add methods to handle the cricket-check sequence and other commands. I will probably do that over the course of the semester.

For now, here is a sample driver program and CricketComm:

```
public class PassingDriver
{
    private static final void sleep(int tenths)
    {
        try {
            Thread.sleep(tenths*100);
        } catch (InterruptedException e) {}
    }

    public static void main(String[] args)
    {
        try {

            CricketComm cricket = new CricketComm();

            cricket.enableAsyncReceive(true);

            sleep(20);
            cricket.send(0x29);
            sleep(20);
            cricket.send(0x19);
            sleep(100);
            cricket.close();

        } catch (Exception e) {
            e.printStackTrace(System.out);
        }
    }
}
```

10

20

```
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.TooManyListenersException;
import javax.comm.CommPortIdentifier;
import javax.comm.NoSuchPortException;
import javax.comm.PortInUseException;
import javax.comm.SerialPort;
import javax.comm.SerialPortEvent;
import javax.comm.SerialPortEventListener;
import javax.comm.UnsupportedCommOperationException;

public class CricketComm implements SerialPortEventListener
{
    private static final String hex(int x)
    {
        return Integer.toHexString(x);
    }

    private SerialPort fPort;
    private InputStream fIn;
    private OutputStream fOut;

    CricketComm()
        throws NoSuchPortException,
               PortInUseException,
               UnsupportedCommOperationException,
               TooManyListenersException,
               IOException
    {
        CommPortIdentifier id = CommPortIdentifier.getPortIdentifier("COM2");
        fPort = (SerialPort) id.open("Passing", /*timeout=*/ 0);
        fPort.addEventListener(this);
        fPort.setSerialPortParams(/*baudrate=*/ 9600,
                                   SerialPort.DATABITS_8,
                                   SerialPort.STOPBITS_1,
                                   SerialPort.PARITY_NONE);

        fIn = fPort.getInputStream();
        fOut = fPort.getOutputStream();
    }

    void send(int value)
        throws IOException
    {
        fOut.write(value);
        System.out.println("Send 0x"+hex(value));
    }
}
```

```
int receive() 50
    throws IOException
{
    int value = fIn.read();
    System.out.println("Recv 0x"+hex(value));
    return value;
}

void close() 60
{
    fPort.close();
}

void enableAsyncReceive(boolean flag)
{
    fPort.notifyOnDataAvailable(flag);
}

public void serialEvent(SerialPortEvent event) 70
{
    try {
        if (event.getEventType() == SerialPortEvent.DATA_AVAILABLE)
            System.out.println("Recv 0x"+hex(fIn.read()));
    } catch(IOException e) {
        System.out.println("Async recv failed");
    }
}
}
```
